UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Diseño e implementación de un controlador hardware para un sistema inestable

Autor/es

ROOSBEL VINICIO GUAYA VEGA

Director/es

JAVIER ESTEBAN VICUÑA MARTÍNEZ y JAVIER RICO AZAGRA

Facultad

Escuela Técnica Superior de Ingeniería Industrial

Titulación

Grado en Ingeniería Electrónica Industrial y Automática

Departamento

INGENIERÍA ELÉCTRICA

Curso académico

2019-20

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

TRABAJO DE FIN DE GRADO

TITULACIÓN: GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

CURSO: 2019/2020 **CONVOCATORIA:** SEPTIEMBRE

TÍTULO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR HARDWARE PARA UN SISTEMA INESTABLE

AUTOR: ROOSBEL VINICIO GUAYA VEGA

DIRECTOR/ES: JAVIER ESTEBAN VICUÑA MARTÍNEZ Y JAVIER RICO AZAGRA

DEPARTAMENTO: INGENIERÍA ELÉCTRICA

SUMMARY

This Project consist on the design of a controller for a two-wheeled robot. The design will be done on a development board of the SoC Zynq 7007s, the MiniZed. From the MPU 6050 sensor will be obtained the values from its gyroscope and accelerometer. These values will be passed to a complementary filter and from it the inclination will be obtained. For the motor control, a dual motor driver will be used. A design on the programable logic will be made to control the motor driver from the MiniZed and to obtain the encoder values from the wheels.

The mathematic model of the robot will be obtained. A controller will be designed and then implemented on the real system and then adjusted based on the behavior of the robot.

RESUMEN DEL PROYECTO

Este proyecto consiste en el desarrollo de un controlador para un robot de dos ruedas para que se auto equilibre. El diseño se hará en una placa de desarrollo del dispositivo SoC Zynq 7007s, la MiniZed. La orientación se obtendrá con un módulo MPU 6050 que incluye un giroscopio y un acelerómetro. El ángulo de inclinación se obtendrá haciendo pasar los valores de aceleración y de velocidad angular por un filtro complementario. Para el control de los motores se usará un driver capaz de dirigir los dos motores y se desarrollarán periféricos hardware para controlar el driver de los motores y obtener la información de los dos encoder incrementales de los motores.

Se obtendrá además el modelo matemático del robot al cual se intentará generar un controlador, que luego será ajustado cuando se lo implemente en el sistema real basándose en su comportamiento.

UNIVERSIDAD DE LA RIOJA

TRABAJO DE FIN DE GRADO

TITULACIÓN: GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

CURSO: 2019/2020 **CONVOCATORIA:** SEPTIEMBRE

TÍTULO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR HARDWARE PARA UN SISTEMA INESTABLE

AUTOR: ROOSBEL VINICIO GUAYA VEGA

DIRECTOR/ES: JAVIER ESTEBAN VICUÑA MARTÍNEZ Y JAVIER RICO AZAGRA

DEPARTAMENTO: INGENIERÍA ELÉCTRICA

ÍNDICE GENERAL

ÍNDICE DE PLANOS

UNIVERSIDAD DE LA RIOJA

TRABAJO DE FIN DE GRADO

TITULACIÓN: GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

CURSO: 2019/2020 **CONVOCATORIA:** SEPTIEMBRE

TÍTULO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR HARDWARE PARA UN SISTEMA INESTABLE

AUTOR: ROOSBEL VINICIO GUAYA VEGA

DIRECTOR/ES: JAVIER ESTEBAN VICUÑA MARTÍNEZ Y JAVIER RICO AZAGRA

DEPARTAMENTO: INGENIERÍA ELÉCTRICA

1 MEMORIA

1.1 HOJA DE IDENTIFICACIÓN

1.1.1 TÍTULO DEL PROYECTO

Diseño e implementación de un controlador hardware para un sistema inestable

1.1.2 PETICIONARIO

Nombre: Universidad de La Rioja, Escuela Técnica Superior de Ingeniería Industrial

Dirección: Calle Luis de Ulloa 4, 26004 Logroño, La Rioja

Teléfono: 941 299 221

1.1.3 DATOS DEL AUTOR DEL PROYECTO

Nombre del autor: Roosbel Vinicio Guaya Vega

DNI: 17495176L

Estudios: Grado en Ingeniería Electrónica Industrial y Automática

Dirección: Calle Pilarte 22

Código Postal: 26500

Teléfono de contacto: 628 69 06 13

Correo electrónico: viguaya@unirioja.es

1.1.4 RESPONSABLES DE LA TUTORÍA DEL PROYECTO

Nombre: Javier Esteban Vicuña Martínez

Ubicación: Calle Luis de Ulloa 4, Edificio de Departamental, Despacho 111

Teléfonos de contacto: 941 29 94 84

Correo electrónico: javier.vicuna@unirioja.es

Nombre: Javier Rico Azagra

Ubicación: Calle Luis de Ulloa 4, Edificio de Departamental, Despacho 106

Teléfonos de contacto: 941 29 94 79

Correo electrónico: javier.rico@unirioja.es

1.2 OBJETO

En este proyecto se intenta hallar una solución al problema de péndulo invertido materializado en el diseño y desarrollo del control de estabilidad de un robot de un eje y doble rueda, que se auto equilibra. Para ello se usará una tarjeta de desarrollo, en concreto la tarjeta MiniZed, que dispone de un dispositivo FPGA de Xilinx de la familia Zynq 7007, que incorpora un procesador ARM CORTEX A9 integrado.

Otros objetivos adicionales que contempla este proyecto son:

- Analizar y poner en práctica el flujo de desarrollo de sistemas SoC.
- Familiarizarse con los programas de desarrollo de sistemas SoC de Xilinx, concretamente Vivado y SDK.
- Examinar y familiarizarse con la documentación que provee Xilinx sobre las ZYNQ-7000, recopiladas en la aplicación DocNav.
- Entender el protocolo AXI y desarrollar periféricos con capacidad de comunicación AXI4 para el módulo que controla los motores y para los encoder.
- Desarrollar una aplicación de comunicación I2C con el MPU 6050 para obtener los datos del acelerómetro y el giroscopio.
- Manejar las interrupciones y periféricos del CPU ARM CORTEX-A9, y programarlas en SDK.

En cuanto a la parte del robot:

- Entender el funcionamiento del filtro Complementario para obtener información sobre la orientación.
- Entender el funcionamiento del sensor MPU 6050 y ser capaz de obtener la inclinación, usando un filtro complementario.
- Conectar los periféricos creados en la parte de lógica programable (PL) para controlar el driver de los motores TB6612FNG y para recibir información de los encoder.

En cuanto a la parte del desarrollo del controlador:

- Hallar las ecuaciones que definen el modelo matemático del robot.
- Simular el comportamiento del robot en Simulink.
- Diseñar un controlador para el robot.
- Ajustar los valores del PID para mejorar su comportamiento.

1.3 ALCANCE

Este proyecto contempla el desarrollo de las partes descritas en el apartado 1.2 de este documento y contempla al menos la implementación de ciertas partes del hardware de control del mismo. Centra su interés en documentar cómo funcionan los dispositivos Zynq y cómo se programan y desarrollan sistemas de control con este tipo de tecnología. Además, se pretende que este proyecto sirva como referencia para aquellos alumnos que estén empezando a manejar los sistemas SoC y las Interfaces AXI, pues se ha puesto especial esfuerzo en explicar el funcionamiento de ambos.

1.4 ANTECEDENTES

El robot del que se propone su diseño se fundamenta en el funcionamiento de un péndulo invertido con dos ruedas. Se han llevado a cabo muchas investigaciones sobre el “robot equilibrista”. Una de sus ventajas es que necesitan menos energía y tienen mejor maniobrabilidad en comparación con los robots de cuatro ruedas.

Debido a la no linealidad de la estructura del robot, las dinámicas del sistema son complejas lo cual dificulta controlar sus movimientos.

El péndulo invertido es normalmente descrito como una barra rígida unida en un extremo por un pasante a un carro. El carro puede desplazarse a lo largo de una vía larga y recta con una fuerza como entrada al sistema.

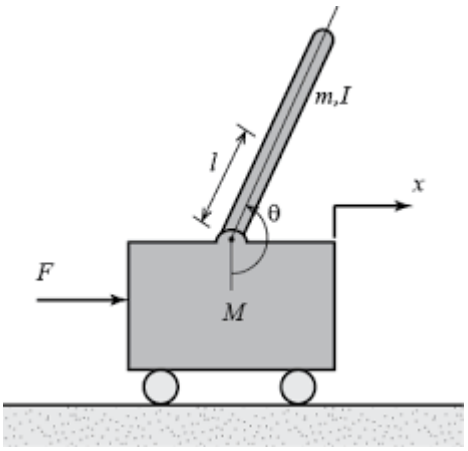


Figura 1-1 Péndulo invertido unido a un carro

El péndulo en sí no es actuado, es libre de girar alrededor del pasador. Tiene dos puntos de equilibro, como se ve en la figura anterior, uno es cuando $\theta = 0$ y otro cuando $\theta = \pi$. Pero es muy difícil que se quede en la posición de $\theta = \pi$ debido a que, a la mínima derivación de posición, el péndulo tiende a caer. Solo si se actúa debidamente el carro y usando la fuerza de reacción del péndulo en contra del carro puede controlarse su posición.

Esta característica de operación corresponde con el modelo de un péndulo invertido. Es un sistema en el que se actúa indirectamente, es decir, el péndulo sólo será controlado ejerciendo fuerza en el objeto secundario de estudio: el carro.

El reto de implementar un controlador en el mundo real ha atraído la atención de muchos investigadores alrededor del mundo, para citar a unos pocos: JOE diseñado en el Industrial Electronic Laboratory [1] o uBot5 diseñado en el Laboratory for Perceptual Robotics. [2]

Figura 1-2 JOE

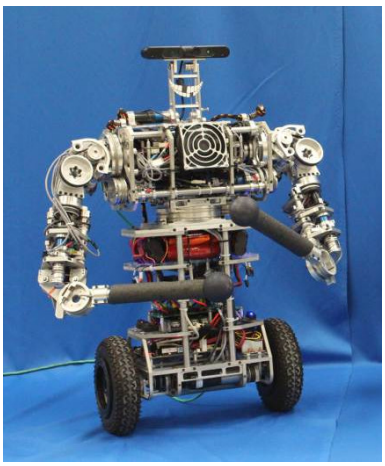


Figura 1-3 uBot-5

Se ha desarrollado una aplicación comercial basada en el control de este sistema, por Dean Kamen en 2001, cuando desveló su nuevo sistema de transporte, los Segway. Estos son en principio un péndulo con dos ruedas, diseñado para balancearse y mantener una postura recta y para transportar a su pasajero a un nuevo lugar.

Figura 1-4 Dean Kamen en su Segway

Quizá debido a la fama del Segway, los robots de dos ruedas que se auto equilibran se han convertido en un proyecto muy común para los estudiantes.

1.5 NORMAS Y REFERENCIAS

1.5.1 DISPOSICIONES LEGALES Y NORMAS APLICADAS

- **UNE 157001:** 2014 Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico.
- **Real Decreto 842/2002, de 2 de agosto**, por el que se aprueba el Reglamento electrotécnico para baja tensión.

1.5.2 PROGRAMAS DE CÁLCULO

- Xilinx Vivado
- Xilinx SDK
- Sigasi
- Matlab
- HScope
- PulseView
- Visual Studio Community
- Solid Works y Visualice

1.6 Bibliografía

[1] F. GRASSER, A. D'ARRIGO, S. COLOMBI y A. RUFER, «JOE: A Mobile, Inverted Pendulum».

[2] S. R. Kuindersma, E. Hannigan, D. Ruiken y R. A. Grupen, «Dexterous Mobility with the uBot-5 Mobile Manipulator».

[3] ARM, AMBA AXI And ACE Protocol Specification.

[4] Xilinx, *UG585 Zynq-7000 Soc Technical Reference Manual,* 2019.

[9] Xilinx, *Zynq-7000 SoC Software Developers Guide,* 2019.

[10] Xilinx, *Vivado Design Suite User Guide: Embedded Processor Hardware Design,* 2019.

[11] Xilinx, *Vivado Design Suite User Guide: Creating and Packaging Custom IP,* 2019.

[12] Xilinx, «Zynq-7000 SoC Product Selection Guide».

1.7 OTRAS REFERENCIAS

Páginas web

[5] AVNET,«Element14,»[Enlínea].Available: https://www.element14.com/community/docs/DOC-94385/l/minized-hardware-technical-training-2019-lesson-1?ICID=minized-zynchardware-training-video. [Último acceso: 02 09 2020].

[6] M. Sadri, «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=0Dt8rWJdiJo&t=6s. [Último acceso: 02 09 2020].

[7] «Sparkfun,» [En línea]. Available: https://www.sparkfun.com/products/14451. [Último acceso: 02 09 2020].

[8] «aalco,» [En línea]. Available: http://www.aalco.co.uk/datasheets/Aluminium-Alloy-5754-H22-Sheet-and-Plate_153.ashx. [Último acceso: 1 09 2020].

1.8 DEFINICIONES Y ABREVIATURAS

1.8.1 DEFINICIONES

System on Chip (SoC)	Dispositivo formado por un CPU y una FPGA
SRAM	(Static RAM) Es una memoria de acceso aleatorio a memoria que conserva su valor siempre que sea alimentada.
IOP	Input-Output Processor. Es un procesador con capacidad de acceder la memoria dirceamente. El IOP es similar a un CPU, excepto en que solo maneja los detalles del procesamiento de señales de entrada y salida (I/O). El IOP puede recoger y ejecutar sus propias instrucciones, pero estas instrucciones son solo usadas para el manejo de las transferencias I/O.
SHIELD	Dispositivo que se conecta sobre otra placa, ofreciendo compatibilidad de pines. Suelen tener incrustados sensores que van conectados con los pines a la placa de desarrollo, permitiendo que no sea necesario ni una protoboard ni soldar.
Protoboard	Dispositivo con carriles que permite conectar distintos dispositivos sin necesidad de soldar.
Soft Core	Un dispositivo programado en lógica programable
Hard Core	Un dispositivo físico que va unido a la FPGA
Drift	Tendencia a variar las medidas de un sensor y alejarse de la medida real a medida que pasa el tiempo
Bias	Tendencia de un sensor a dar un valor constante aun cuando debería dar un valor cero
CC	Corriente continua
DC	Corriente continua

1.8.2 ABREVIATURAS

Término	Significado
CW	Sentido de las agujas del reloj (Clock Wise)
CCW	Sentido contrario a las agujas del reloj
PL	Lógica Programable
PS	Software Programable, se refiere a la parte de la Zynq que contiene el CPU y los periféricos del CPU.
IP	Propiedad intelectual
OCM	On Chip Memory
CPU	Central Processing Unit
FIFO	First Input First Output
SCU	Snoop Control Unit
ACP	Accelerator Coherency Port
IC	Integrated Circuit

1.9 REQUISITOS DE DISEÑO

1.9.1 CLIENTE

El cliente, La Universidad de La Rioja, solicita el diseño de un robot de dos ruedas que se auto equilibra.

1.9.2 ESTUDIOS REALIZADOS ENCAMINADOS A LA DEFINICIÓN DE LA SOLUCIÓN ADOPTADA

1.9.2.1 Protocolo AXI

Para el desarrollo de periféricos en la lógica programable de la FPGA y que se comuniquen con el CPU es necesario conocer cómo funciona y cómo crear Interfaces AXI. Por ello lleva a cabo un análisis según el protocolo AXI, el cual fue especificado por ARM en "*AMBA AXI And ACE Protocol Specification*".

A continuación, se explicará cómo funciona este protocolo de comunicación:

Cada Interfaz de AXI se compone siempre de dos partes, un AXI Master y un AXI Slave, como se puede ver en la siguiente figura.

Figura 1-5 Esquema básico de una Interfaz AXI

Las transacciones pueden ser tanto de lectura como de escritura, pero el AXI Master siempre inicia la transacción.

- Para las transacciones de lectura tenemos un flujo de datos del AXI Slave al AXI Master.

- Para una transacción de escritura tenemos datos que van desde el AXI Master al AXI Slave.

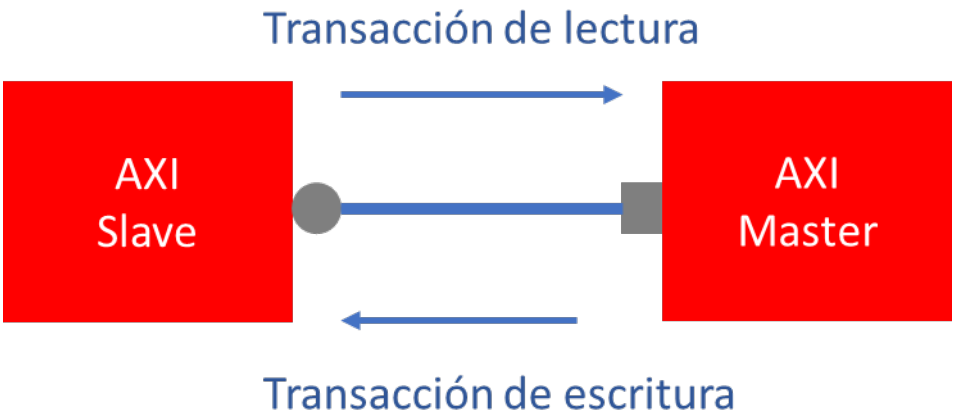


Figura 1-6 Esquema de tipos de transacciones

Existen dos tipos de interfaces AXI denominados AXI Stream y AXI Memory Mapped, que se describen a continuación:

1.9.2.2 AXI STREAM

Este tipo de interfaz es muy útil cuando tenemos bloques que siempre van a estar enviando datos a una sola dirección.

Por ejemplo, se tiene bloques que reciben datos, hacen un tipo de procesamiento en los datos y los pasan al siguiente bloque. Los datos vienen de un ADC, que van a un filtro y los datos son pasados al siguiente módulo. En este caso es muy conveniente este tipo de interfaz.

En este caso, cuando un módulo quiere transmitir datos a otro módulo, no necesita transmitir la dirección pues siempre escribe al mismo y, además, la dirección de datos es siempre unidireccional. Los datos fluyen de un módulo al siguiente y nunca se invierte el sentido.

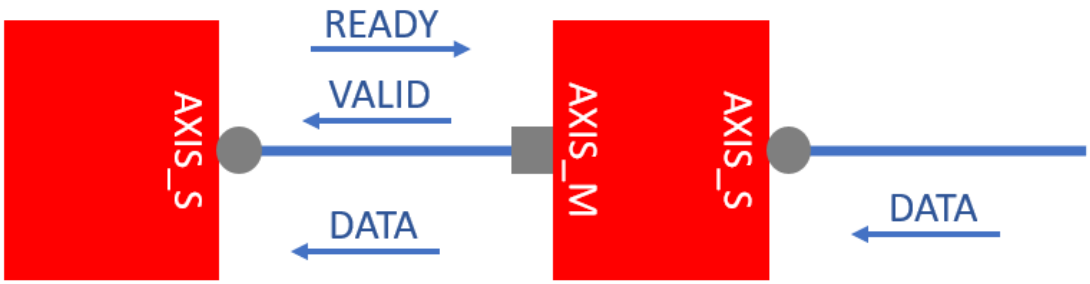


Figura 1-7 Esquema AXI Stream

Una Interfaz AXI STREAM es básicamente un canal WRITE DATA de la interfaz AXI MEMORY MAPPED.

Está compuesto por las señales READY, VALID y DATA

1.9.2.3 AXI MEMORY MAPPED

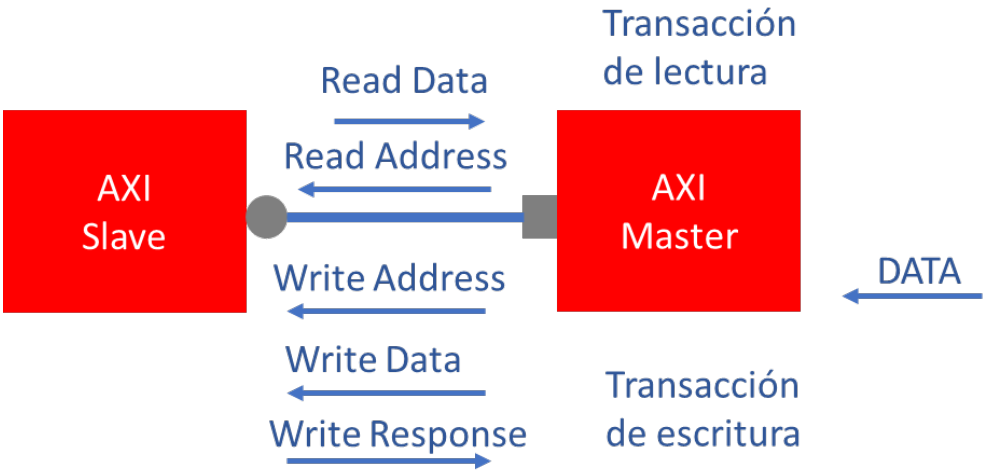


Figura 1-8 Esquema de AXI Memory Mapped

Las conexiones entre AXI MM siempre se realizan con AXI Interconnect que une a los AXI Master y a los AXI Slave. El AXI Interconnect tiene la función de proporcionar a cada AXI Slave un rango de direcciones en el mapa de memoria. También permite conectar AXI Masters y AXI Slaves trabajando en distintos dominios de reloj.

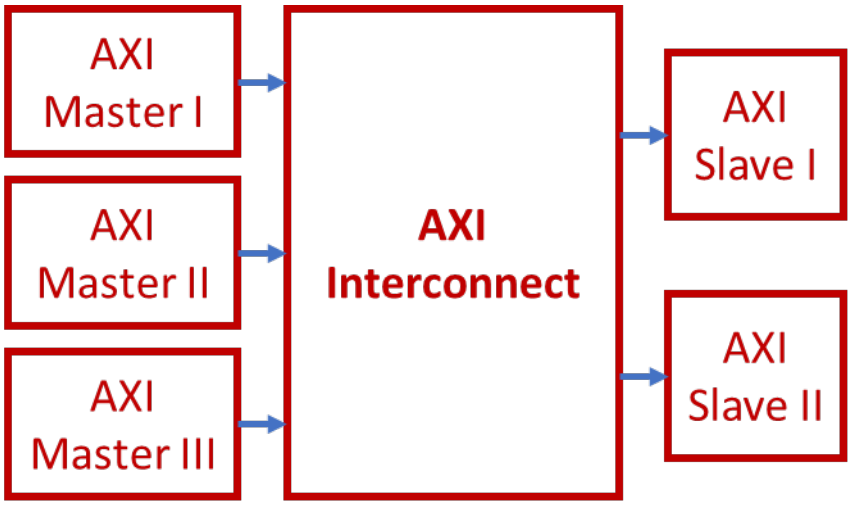


Figura 1-9 Esquema de una conexión AXI MM

Dentro de interfaz AXI MEMORY MAPPED tenemos:

- AXI MM FULL: Que contiene más señales y es más complicado diseñar con él. Pero tiene la ventaja que permite la lectura y escritura en ráfagas, lo cual permite hacer transacciones de

varias palabras. Además de implementar las señales ID que permiten aumentar aún más la velocidad cuando hay varios maestros.

- AXI MM Lite: Es conveniente cuando no requerimos de tanta banda ancha, pues trabaja con un número reducido de señales que permiten solo la transferencia de una palabra por transacción. Su gran ventaja es que es mucho más sencilla de implementar.

La diferencia de tener las señales ID en de las dos Interfaces AXI MM FULL y AXI MM Lite, se hace clara cuando tenemos una configuración como la se indica en la imagen, en la que tenemos varios AXI Masters unidos a un AXI Interconnect, que los une al AXI Slave.

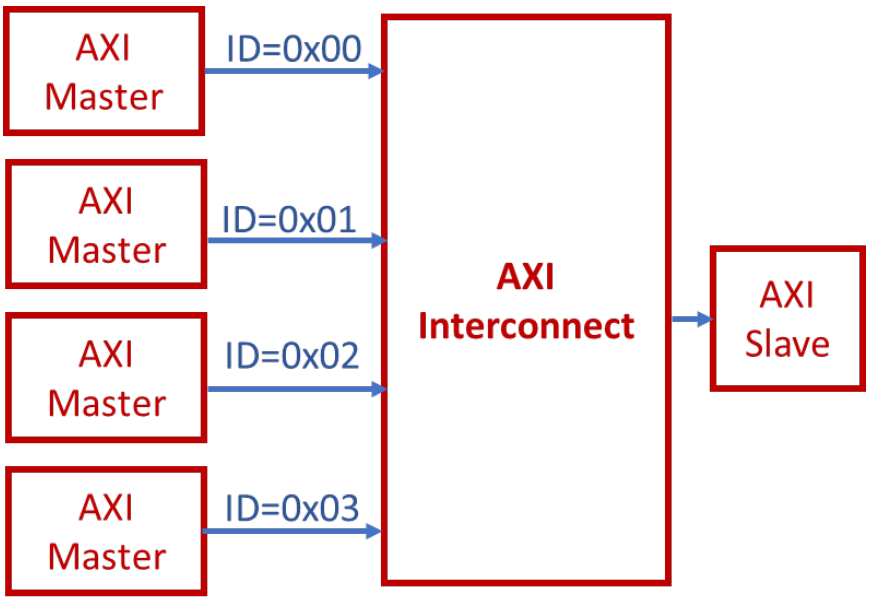


Figura 1-10 Esquema de una conexión AXI MM

Todos de estos AXI Masters están operando y necesitan comunicarse con el AXI Slave.

Si se usa una interfaz AXI Slave Lite, solo se puede realizar una transacción a la vez, ya que el AXI Interconnect no sabría distinguir a cuál de los AXI Master debe escribir la respuesta del AXI Slave si se realizaran múltiples transacciones.

Sin embargo, si se usa una interfaz AXI-Full el AXI Interconnect como puede asignar una ID a cada uno de los AXI Master y el AXI Slave puede indicar para cuál de los AXI Master es la respuesta. Con la ID de transacción se puede realizar:

- Cualquiera de los AXI Master puede requerir el comienzo de una transacción en cualquier momento, no tiene que esperar a que termine la transacción anterior.
- El AXI Slave puede ejecutar respuestas en distinto orden de cómo fueron iniciadas, ya que cada transacción está identificada por una ID.

Estas señales son en el AXI-Full MM:

- **AWID** (En el canal WRITE ADDRESS): Cada vez que el maestro inicia una transacción, él pone un valor en la ID.

- **BID** (En el canal WRITE RESPONSE): Cuando el AXI Slave responde con a esta específica transacción de escritura la señal BID pondrá la misma ID que puso el AXI Master en el AWID cuando estaba escribiendo la dirección a la que está dirigida la transacción.
- La misma operación sucederá cuando se ejecuta una transacción de lectura en las señales **ARID** (En el canal READ ADDRESS) y **RID** (En el canal READ RESPONSE).

En este tipo de interfaz, AXI MEMORY MAPPED, se tiene varios canales que se usaran dependiendo de si vamos a hacer una lectura o una escritura.

La explicación del funcionamiento se hará de una interfaz AXI-Lite MEMORY MAPPED. Esto es porque es el tipo de interfaz que se ha usado para comunicar los periféricos con el CPU y también porque el menor número de señales permite una mejor comprensión de su funcionamiento.

En el siguiente apartado se realizará una explicación completa de todas las señales que se pueden encontrar en una interfaz AXI-FULL MEMORY MAPPED.

Transacción de lectura:

La siguiente figura indica los canales envueltos en la transacción de lectura. Las flechas indican el sentido que tienen los buses de datos en cada uno de los canales.

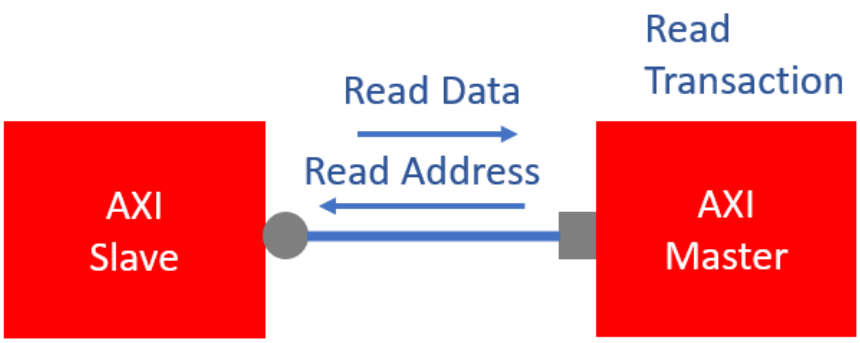


Figura 1-11 Esquema de la transacción de lectura

Si vamos a hacer una lectura de un registro del AXI Slave se sigue el siguiente procedimiento:

El AXI Master debe informar que tiene una dirección de la cual quiere leer, poniendo a nivel alto ARVALID. El AXI Master tiene que asegurarse de tener en el ARADDR el valor correcto de la dirección que quiere leer, para cuando pone a nivel alto ARVALID.

ARVALID y ARADDR son señales del canal READ ADDRESS.

El AXI Slave una vez detecta que la señal ARVALID='1', lachea la señal ARADDR y genera un pulso de duración un S_AXI_ACLK en la señal ARREADY del canal READ ADDRESS.

Durante la duración del pulso en ARREADY, se lleva a cabo la obtención de los datos del registro que se quiere leer y se los coloca en la señal RDATA del canal READ DATA.

Un pulso de S_AXI_ACLK después de la generación de ARREADY, se lacheará RVALID a nivel alto, así como el valor de RRESP. RVALID y RRESP son señales del canal READ DATA.

Con la señal RVALID a nivel alto el AXI Slave informa al AXI Master que tiene datos listos para ser leídos en RDATA. Si señal RRESP tiene un valor de “00” al final de la transacción quiere decir que la transacción ha sido correcta.

El valor de RVALID seguirá a nivel alto hasta que el AXI Master ponga a nivel alto el RREADY.

Transacción de escritura

La siguiente figura indica los canales envueltos en una transacción de escritura. Las flechas indican el sentido que tienen los buses de datos en cada uno de los canales.

Figura 1-12 Esquema básico de una transacción de lectura

Para iniciar la transacción de escritura el AXI Master debe señalar por el canal WRITE ADDRESS que tiene una dirección a la cual quiere escribir (AWVALID=’1’). También debe señalar que tiene una palabra válida en el canal WRITE DATA (WVALID=’1’). Cuando el AXI Master señala esto, debe tener en las señales con la dirección de escritura (AWADDR) del canal WRITE ADDRESS y la señal con el bus de datos a escribir (WDATA) del canal WRITE DATA listos para ser leídos.

Cuando el AXI Slave detecta que ambas señales AWVALID y WVALID están a nivel alto, el AXI Slave procede a lachear la dirección (AWADDR) y genera un pulso de duración de un S_AXI_ACLK en las señales AWREADY y WREADY.

Mientras dura el pulso en AWREADY y WREADY, y solo si la dirección escrita en AWADDR es correcta, se lleva a cabo la escritura de WDATA en el registro direccionado por AWADDR.

Además de la escritura de los registros, mientras dura el pulso de AWREADY y WREADY, se lachea, por parte del AXI Slave, la respuesta a la transacción (BRESP), además de informar al AXI Master que tiene una respuesta a la transacción disponible para ser leída (BVALID=’1’). Las señales BRESP y BVALID están en el canal WRITE RESPONSE. Si la transacción de escritura se ha realizado con éxito la respuesta debe ser BRESP=”00”.

La señal BVALID permanecerá en estado alto hasta que el AXI Master ponga la señal BREADY a nivel alto.

Señales en detalle de los canales para una Interfaz AXI-FULL MEMORY MAPPED

Señales en el canal WRITE ADDRESS

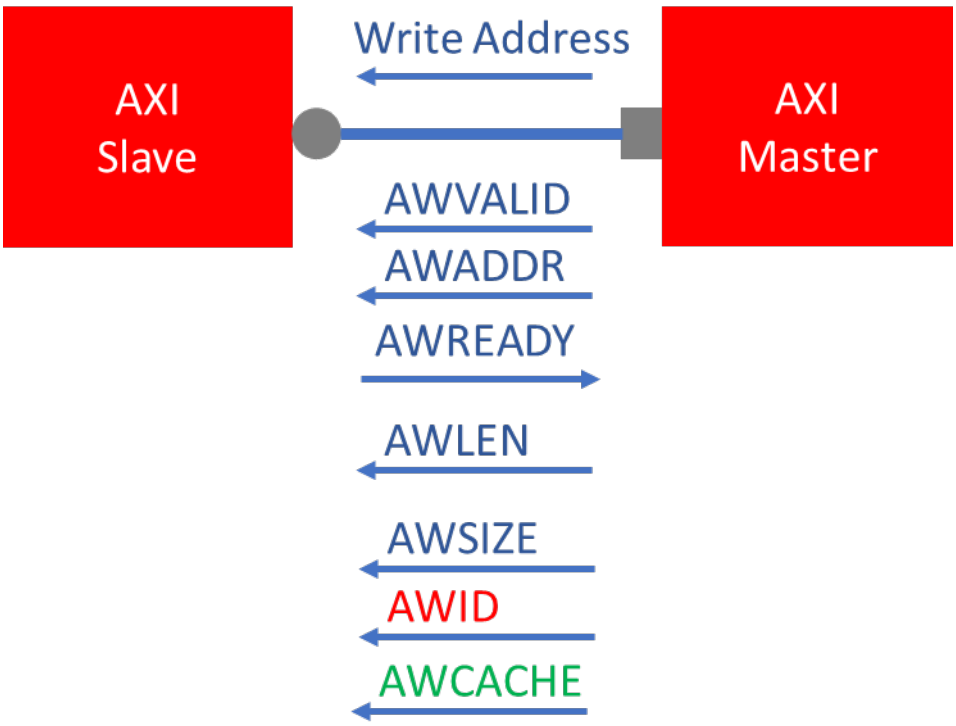


Figura 1-13 Esquema de las señales en el canal WRITE ADDRESS

Las tres primeras señales son las mismas que se usan en la Interfaz AXI STREAM.

AWVALID:(Maestro a esclavo) Indicando que el maestro tiene una dirección y la quiere enviar al esclavo.

AWADDR:(Maestro a esclavo) La dirección en sí.

AWREADY:(Esclavo a maestro) Indica que el esclavo está listo para recibir la señal.

AWLEN:(Maestro a esclavo) Indica el número de palabras por ráfagas que queremos hacer en la transacción. Por ejemplo, si queremos realizar una ráfaga de 8 palabras en este puerto se escribe el número 8(1000'b) y con ello el esclavo esperará 8 palabras de datos cuando se realice la transferencia de escritura.

AWSIZE:(Maestro a esclavo) Indica la anchura en bits de los paquetes que debe esperar el esclavo.

AWID:(Maestro a esclavo) El AXI Maestro es capaz de dar una ID a cada una de las transacciones que realiza. Por ejemplo, tiene que realizar 3 transacciones, a la primera le pondrá un id de 1, a la segunda de 2 y a la tercera de 3.

AWCACHE:(Maestro a esclavo) Nos sirve cuando los AXI Slave tienen funcionalidades de cachear los datos o de meterlos a un buffer y escribirlos después… etc.

El AXI Interconnect tiene unas capacidades adicionales. Tiene un buffer en el que escribir los datos que recibe del AXI Master antes de escribir los datos al AXI Slave. También puede poner en un buffer los datos que recibe del AXI Slave, así cuando otro AXI Master quiere acceder a esa misma dirección, el AXI Interconnect puede responder sin necesidad de acceder al AXI Slave. A través AWCACHE, el AXI Master indica si no le importa que el bloque que está entre el AXI Master y el AXI Slave, use su cache o buffer interno para responder a la transacción de manera más rápida, sin tener que escribir al AXI Slave.

Señales en el canal WRITE DATA

Por este canal el AXI Master transfiere las palabras que quiere escribir al AXI Slave.

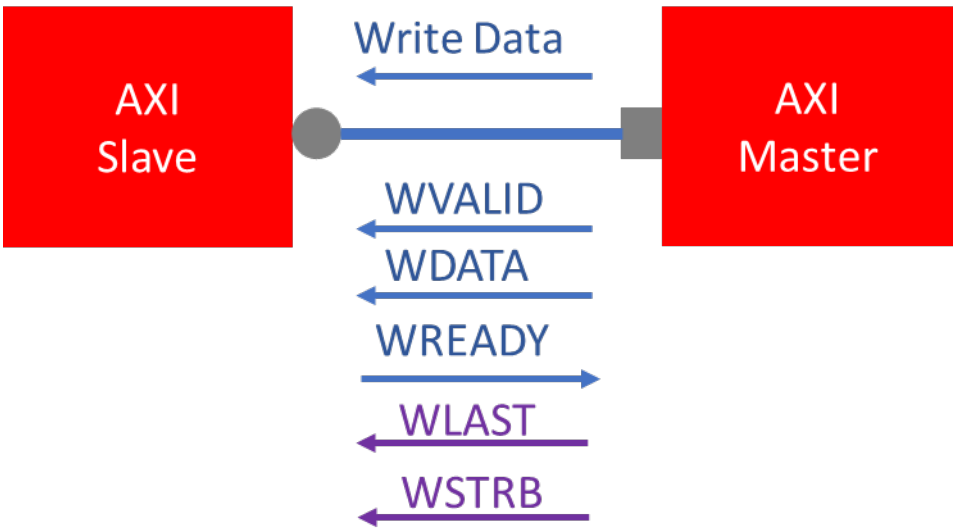


Figura 1-14 Esquema de las señales en el canal WRITE DATA

WVALID:(Maestro a Esclavo) Indica que el AXI Master tiene los datos listos en WDATA para que los reciba el AXI Slave.

WDATA:(Maestro a esclavo) Son los datos en si.

WREADY:(Esclavo a maestro) AXI Slave indica que está listo para recibir la transacción.

WLAST:(Maestro a esclavo) Indica que este dato que estoy enviando es el último de la ráfaga.

WSTRB:(Maestro a esclavo) Indica cuales de los bits de WDATA que ha enviado son válidos y cuáles no. Por ejemplo, tienes un bus de 32 bits, pero solo te interesan los 2 primeros bits, con WSTRB se puede realizar esto.

Señales en el canal WRITE RESPONSE

Por este canal, el esclavo indica los resultados de la transacción.

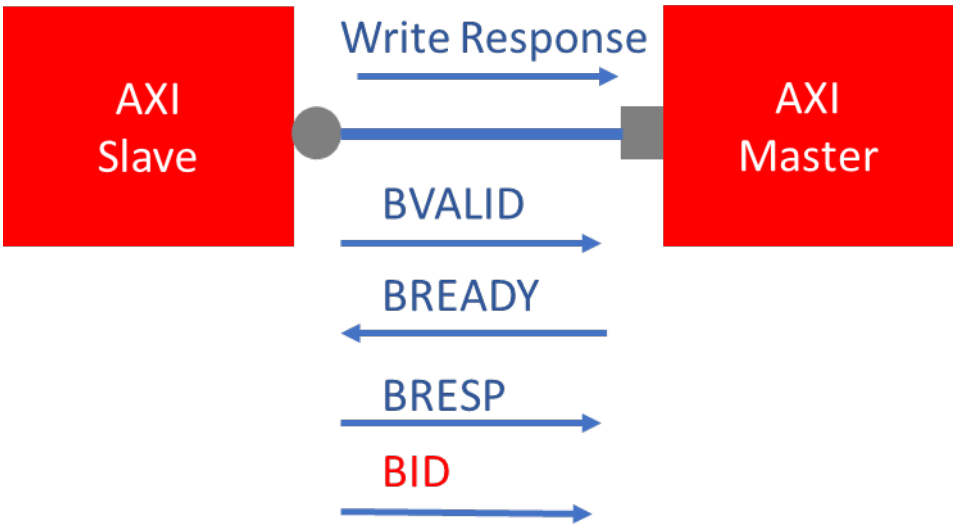


Figura 1-15 Esquema de las señales WRITE RESPONSE

BVALID:(Esclavo al maestro) El AXI Slave indica que tiene una respuesta de cómo ha ido la transacción lista para ser leída

BREADY:(Maestro a esclavo) Con esta señal el AXI Master indica que está listo para recibir la respuesta.

BRESP:(Esclavo al maestro) Es la respuesta en del AXI Slave en cuanto al estado de la transacción de escritura. En la ref: "*AMBA AXI And ACE Protocol Specification*" [3] pag.54 nos encontramos la siguiente tabla con los valores que puede tomar:

RRESP[1:0] BRESP[1:0]	Response
0b00	OKAY
0b01	EXOKAY
0b10	SLVERR
0b11	DECERR

Tabla 1-1 Posibles respuestas en el canal WRITE RESPONSE

BID:(Esclavo al maestro) El AXI Slave puede indicar para cuál de las transacciones está escribiendo la respuesta. Para ello usa el ID de la transacción que el AXI MASTER usó en el canal de WRITE ADDRESS.

Transacción de lectura

Señales del canal READ ADDRESS

Se encarga de transferir la dirección desde la cual se quiere leer, indicando además el tamaño de la lectura que va a ser leído, incluyendo una ID para la transacción.

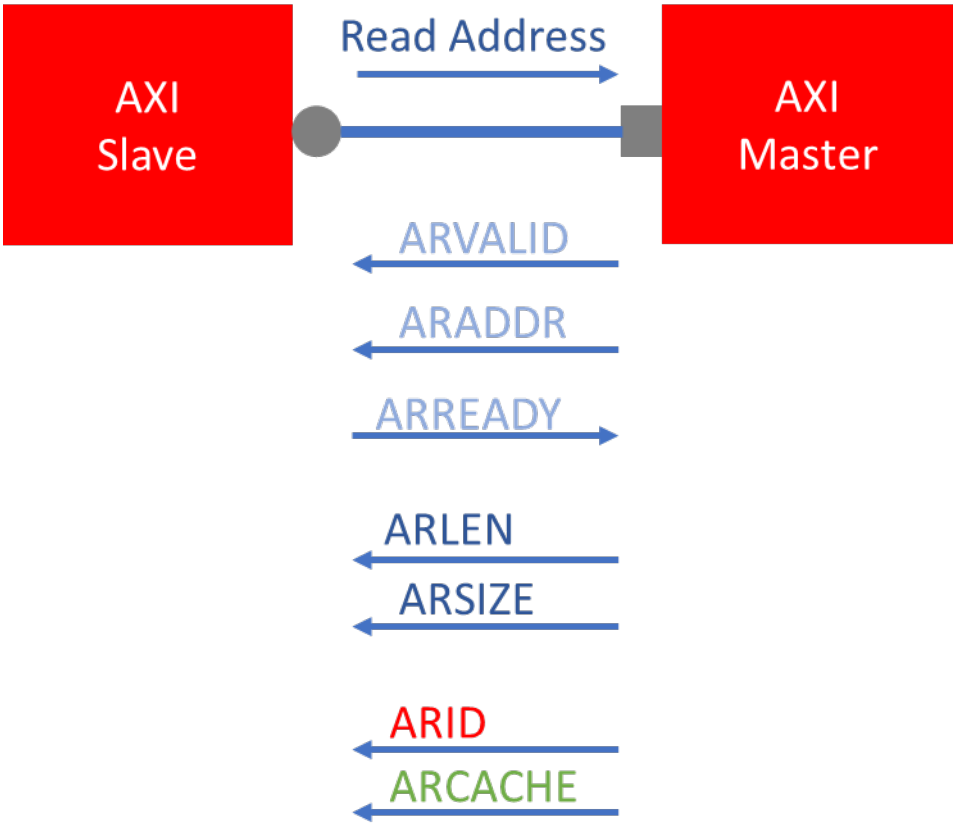


Figura 1-16 Esquema de las señales del canal READ ADDRESS

ARVALID (Maestro a esclavo): Indica que el AXI Master tiene el canal ARADDR listo para ser leído.

ARADDR (Maestro a esclavo): La dirección en sí.

ARREADY (Esclavo al maestro): El esclavo indica que está disponible para recibir la dirección.

ARLEN (Maestro a esclavo): El AXI Master indica con esta señal la cantidad de datos que va a leer del esclavo en una ráfaga.

ARSIZE (Maestro a esclavo): El AXI Master indica el ancho de los datos en bits que van ser leídos. **ARID** (Maestro a esclavo): Con esta señal se identifican las transacciones. **ARCACHE** (Maestro a esclavo): Indica si el AXI Slave puede usar su cache o si tiene que realizar la transacción completa y leerla del destino.

Las señales son prácticamente las mismas que para el canal Write Address, solo cambian los nombres.

Señales del canal READ RESPONSE

Este canal es usado por el AXI Slave para enviar los datos que han sido pedidos por el AXI Master.

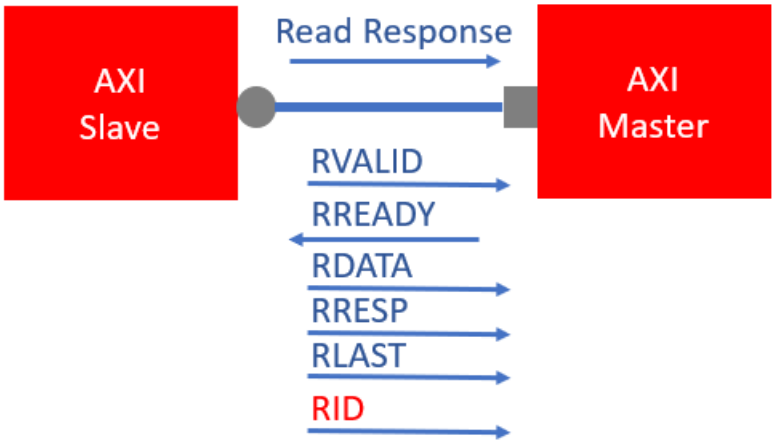


Figura 1-17 Esquema de las señales que tiene el canal READ RESPONSE

RVALID (Esclavo al maestro): Indica que el AXI Slave tiene ahora los datos que quiere el AXI Master y puede comenzar la transacción para enviarlos.

RREADY (Maestro a esclavo): El AXI Master indica con esta señal que está disponible para recibir el RDATA.

RDATA (Esclavo al maestro): Es el dato en sí. Es el dato que tenía el AXI Slave internamente para esa específica dirección.

RRESP (Esclavo al maestro): El AXI Slave indica si esta transacción ha sido correcta o no. Algunas veces el AXI Master envía una petición de lectura al AXI Slave, pero por alguna razón esta dirección no es correcta, o, por ejemplo, el AXI Slave tiene que realizar una lectura a uno de sus submódulos y por alguna razón esta lectura falla.

El AXI Slave tiene que enviar por RRESP el código de si se ha llevado bien a cabo la transacción o si ha fallado.

RLAST (Esclavo al maestro): El AXI Slave indica que el dato que está siendo enviado es el último dato de la transacción tipo ráfaga.

RID (Esclavo al maestro): El AXI Slave indica con esta señal para cual ID de las operaciones de lectura está realizando esta respuesta.

Señales adicionales que pueden estar presentes en una Interfaz AXI MEMORY MAPPED

La mayoría de las veces no son usadas, pero según la documentación también están incluidas en el protocolo.

- **Interfaz de bajo consumo:**
  Se utilizan cuando el periférico puede cambiar entre varios estados de funcionamiento. Con esta interfaz un CPU pueden despertar a los periféricos cuando lo necesite.
- **Tipo de protección:**
  Cuando se tiene un sistema para el cual la seguridad de las transacciones entre distintas partes del sistema es muy importante. Este conjunto de señales sirve para indicar el nivel de seguridad que necesita el sistema.

- **Transacciones para QoS (Quality of Service)**
  A veces, cuando se está realizando varias transacciones entre un AXI Master y un AXI Slave o un AXI master y una red de AXI Slaves, pero las cuales no son del mismo nivel de importancia. Estas señales sirven como el mecanismo para indicar cuál transacción es más importante y debe ser respondida más rápidamente y qué transacciones no son tan importantes y pueden ser respondidas después.
  Con estas señales se puede indicar el nivel de prioridad de las transacciones que estas llevando a cabo.
- **Identificadores de región (Region Identifiers):**
  Usando el AXI MEMORY MAPPED un periférico AXI puede ocupar varias regiones en el mapa de memoria, en vez de una región de memoria continua.
  Tradicionalmente, un periférico tiene asignado una región continua en el mapa de memoria, pero a través del estándar AXI4, se tiene la posibilidad de tener distintas regiones mapeadas a un mismo periférico AXI.
  Los identificadores de región (Region Identifiers) son un conjunto de señales que permiten diseñar periféricos con esta característica especial.

- **Señales definidas por el usuario.**
  El diseñador puede generar sus propias señales, y con ellas transferir en transacciones datos adicionales que no podrían ser transferidos en una interfaz AXI normal.
  Por ejemplo, en si en una aplicación de procesamiento de video se quiere indicar que el dato que estoy se está enviando ahora es el primer fotograma.

1.9.2.4 Arquitectura de la ZYNQ-7000 SoC

Para desarrollar programas en la parte de software programable (PS) y diseñar hardware en la lógica programable (PL), es necesario llevar a cabo un análisis de la arquitectura de la Zynq. Una vez se ha completado el análisis se habrá respondido a las siguientes preguntas:

- Cuáles son las partes de una ZYNQ y cómo se relacionan entre sí.

- Qué opciones de comunicaciones hay entre el PL y PS, y en cual debe usarse dependiendo de lo que se quiera hacer.

Arquitectura del dispositivo Xilinx Zynq.

La Zynq está divida en dos partes, el PS (Programabe System) y PL (Programable Logic).

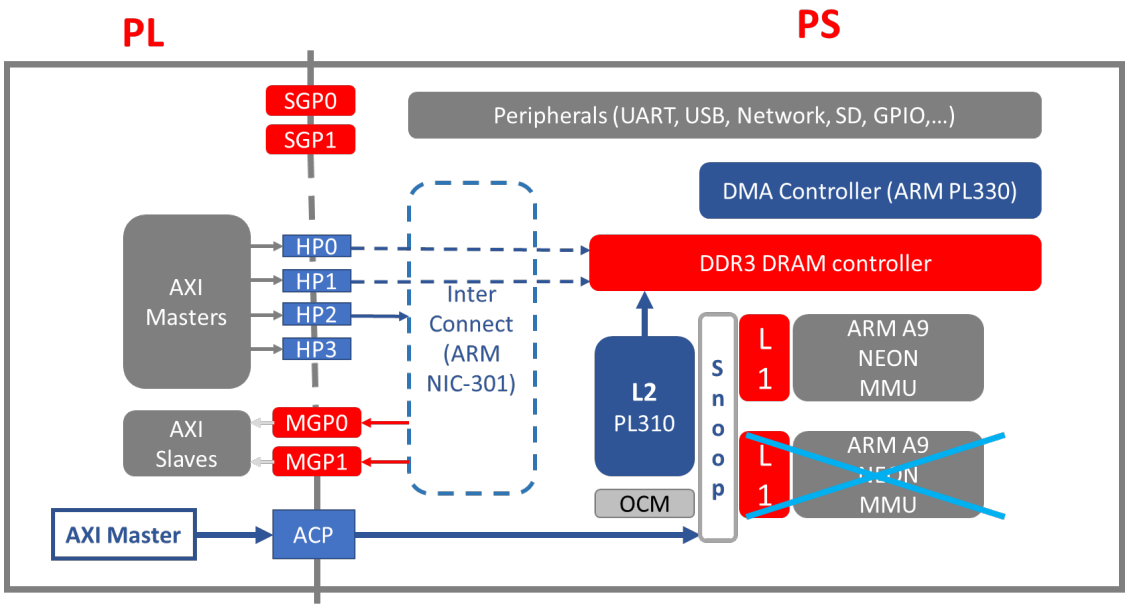


Figura 1-18 Arquitectura Zynq-7007s SoC

El PL es básicamente una FPGA donde podemos programar los periféricos y la lógica programable que queramos. La ZYNQ gracias al ACCELERATOR COHERENCY PORT (ACP), permite que se programen aceleradores hardware en el PL que hagan uso de las cache L1 y L2 del CPU para realizar intercambios de información mucho más rápida y de manera más eficiente energéticamente.

También gracias a la gran cantidad de puertos de comunicación entre el PL y PS, se contempla un ancho de banda de hasta 100Gbps.

El PS contiene un ARM CORTEX A9. La especificación de la Zynq puede contener hasta dos ARM A9, pero el modelo usado por la MiniZed es el 7007s solo tiene uno. Por ello en el esquema de la Figura 1.18 se ha incluido, pero se tachado el segundo CPU para dejar claro que la mayoría de las Zynq tienen y han tenido dos núcleos y no ha sido hasta hace poco que Xilinx ha sacado a la venta modelos de bajo coste con un solo núcleo.

El CPU ARM A9 está conectado a una cache L1 y una cache L2.

La SNOOP CONTROL UNIT se encarga de que haya coherencia entre los datos almacenados en la cache L1 y la L2 del CPU y que si hubiera otro procesador, que no haya conflictos entre datos que sean continuamente accedidos por los dos procesadores y que posiblemente estén en sus respectivas cache L1.

El PS contiene además hardware específico para funciones como UART, USB, ETHERNET, SD, I2C, GPIO. Además, contiene un DMA CONTROLLER que puede ser usado para transferencias de memoria grandes o muy rutinarias y así ayudar al CPU de ser ralentizado.

El PS también contiene un DDR3 DRAM CONTROLLER, que se encarga de gestionar las lecturas y escrituras a la memoria DDR3. En la MiniZed, como se puede ver el esquema de la placa en el ANEXO IV - IV del documento ANEXOS, se ha conectado a una DDR3L de 512MB.

Además, existe una memoria SRAM On-Chip de 256KB (OCM). Esta memoria es volátil, y debe ser programada cada vez que se da alimentación a la placa.

Como se puede ver en la Figura 1.19, hay un Interconnect que conecta el CPU, el DMA CONTROLLER, AXI Masters del PL, IOP Masters con la OCM, DDR3 DRAM CONTROLLER, IOP Slaves y AXI Slaves en el PL.

Para comunicarse con el exterior, Xilinx Vivado permite a los distintos bloques hardware como el I2C, la UART, USB… conectarse al exterior usando el MIO o el EMIO. Cada opción tiene sus ventajas, pero aquí discuto unas diferencias y unos apuntes muy específicos que han sido útiles en el desarrollo del proyecto, más información se puede obtener del documento U585 ZYNQ-7000 TRM [4].

- El MIO tiene 32 pines y el EMIO tiene 64 pines.
- Algunos periféricos del PS solo pueden conectarse por los pines MIO y si se conectan por EMIO, se ven reducidas sus funcionalidades, como puede ser la velocidad máxima a que pueden trabajar.
- En la MiniZed todos los pines MIO están soldados de la ZYNQ y no se pueden acceder físicamente a ellos. Por ejemplo, los pines MIO 52 y 53 (PS_MIO52_501, PS_MIO53_501 en el Plano 2 del documento PLANOS) están conectados a los leds ROJO y VERGE respectivamente.
- La interfaz I2C o UART pueden ser conectadas por el EMIO pero deben ser además incluidas en el archivo CONSTRAINS.
- Las señales del PL saldrán por los pines EMIO.
- Para configurar los módulos hardware del PS, en Vivado se debe instanciar el bloque ZYNQ7 Processing SYSTEM en IP Integrator y permite modificar ahí cómo se quiere que se inicialice el PS, qué bloques hardware PS activar y a qué pines MIO o si van a ser dirigidos al EMIO.

<u>Interfaces AXI de comunicación entre el PL y PS:</u>

Las siguientes interfaces son AXI MEMORY MAPPED, aunque se pueden usar bloques para tratarlos como AXI STREAM Interfaces.

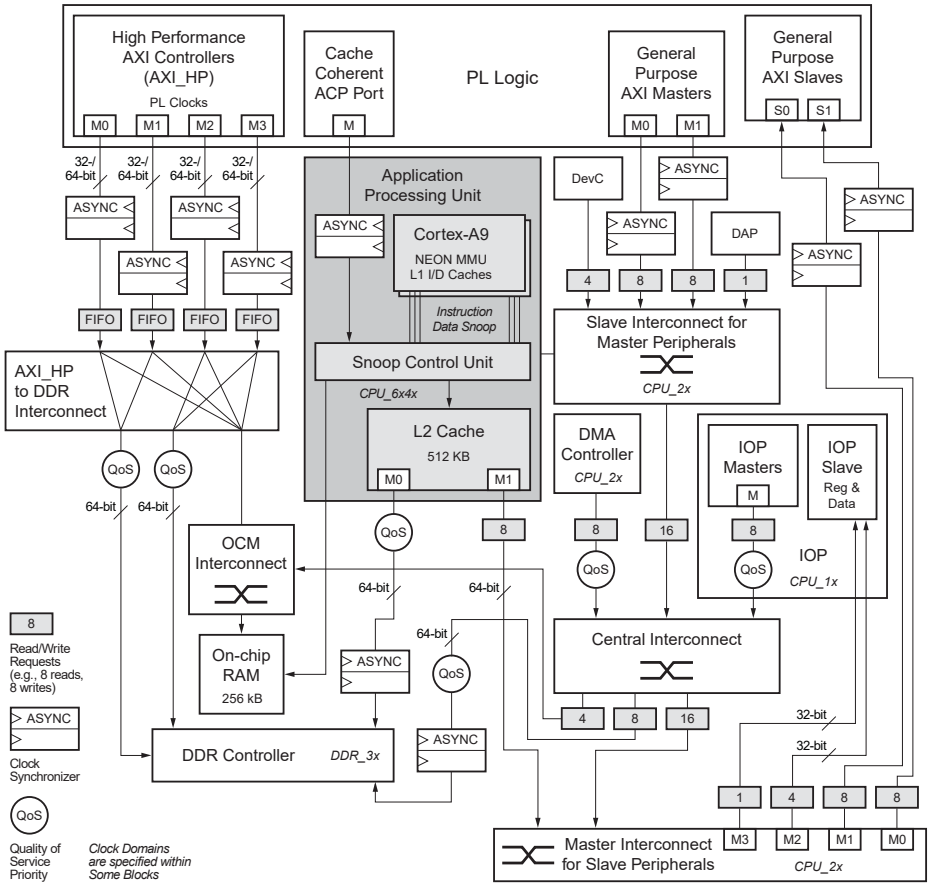


Figura 1-19 Puertos de comunicación entre el PL y PS

AXI HIGH PERFORMANCE PORT (AXI_HP):

Hay cuatro puertos HP. Pueden ser de 32 o 64 bits, y la lógica programada en el PL es siempre el maestro. Estos puertos tienen una gran velocidad de transferencia e incluyen FIFOs que aceptan lecturas y escrituras tipo ráfaga.

Estos puertos como se puede ver en la Figura 1.19, permiten el acceso a la memoria DDR, así como a la OCM.

GENERAL PURPOSE AXI MASTERS (MGP):

Son puertos de 32 bits, que permiten al CPU comunicarse con la lógica programada en la PL. Como se puede ver en la Figura 1.19, gracias al CENTRAL INTERCONNECT, también permiten al DMA CONTROLLER y a los periféricos conectados a los SGP, acceder a la lógica del PL.

Los puertos MGP son muy importantes puesto que son los únicos puertos maestros del PS. Son también la única manera en que el DMA CONTROLLER pueda realizar transacciones de lectura y escritura a la lógica que se ha desarrollado en el PL.

Cuando el procesador ARM inicie una transacción de lectura o escritura a un rango de memoria específico, la transacción aparecerá en estos puertos GENERAL PURPOSE AXI Master(MGP) y luego

será transferido a los módulos esclavos que se han desarrollado en el PL, los cuales deberían proveer una respuesta adecuada que será transferida al procesador ARM.

Hay dos puertos GP0 y GP1. Si se mira el mapa de memoria del ANEXO III, cada puerto GP tiene un rango de memoria de 1GB donde pueden mapear los periféricos diseñados en el PL.

En concreto para el GP0 el rango es: 0x4000_0000 a 7FFF_FFFF.

Para el GP1 el rango es: 0x8000_0000 a 0xBFFF_FFFF

ACCELERATOR COHERENCY PORT (ACP):

Es muy similar a los Puertos HP. Su anchura es 64 bits. Es necesario conectarse al ACP con una Interfaz AXI Master, para poder iniciar transacciones de lectura y escritura.

La principal diferencia con los puertos HP, es que está conectado directamente con la SNOOP CONTROL UNIT (SCU). La SCU está conectado directamente a las caches del CPU, L1 cache, L2 cache.

Cada vez que la transacción es iniciada por el AXI Master en el PL, las cache son chequeadas primero por el dato requerido. En ellas se buscará por la específica dirección física por la cual se ha iniciado la transacción y si una instancia de ese dato está disponible la transacción será respondida con el dato en las caches. Si el dato requerido está disponible, el puerto ACP permite transmisiones más rápidas, más eficientes energéticamente a la lógica que se ha desarrollado en el PL.

Si para la transacción que se ha iniciado, no hay una correspondiente instancia en las caches, la transacción será dirigida a la memoria DRAM, lo cual añade una latencia, que puede degradar la eficiencia de la aplicación.

El ACP debe usarse cuando se tiene una aplicación en la que se quiere que el CPU colabore con un acelerador hardware implementado en el PL, en ese caso el puerto ACP puede ser muy útil. En ese caso, el CPU y el acelerador hardware pueden usar las caches del CPU para compartir datos, esto permite una transferencia muy rápida y con muy poco consumo.

Slave General Purpose Ports (SGP):

Son puertos esclavos del PS. Con ellos los módulos que tengamos en el PL pueden iniciar transacciones a los periféricos en el PS.

Su anchura es de 64bits.

1.9.2.5 Crear una Interfaz AXI Slave

Para el desarrollo del proyecto, ha sido necesario investigar cómo se crean en concreto Interfaces AXI Slave Lite en el entorno de Vivado.

Crear una Interfaz AXI Slave en un periférico que se diseñe en la lógica programable es importante pues se necesita una Interfaz AXI MEMORY MAPPED para recibir órdenes de lectura y escritura por parte del CPU a través de los puertos MGP0 o MGP1.

Para implementar esta interfaz en el periférico se pueden usar varios métodos.

- Escribir tu propia interfaz AXI Slave en RTL.

  Para esto es necesario leerse la especificación AXI de ARM [3] y besándose en la especificación desarrollar la lógica que permitirá al módulo enviar y recibir información a través de AXI.

  Esta opción requiere mucho tiempo de desarrollo y lo desarrollaría alguien muy experimentado.

- Las unidades Xilinx IPIF
  Cada una de estas unidades están especializada en realizar una operación concreta de transferencia de datos.
  PG155: AXI4 Slave Lite IPIF
  PG158: AXI4 Slave Burst IPIF

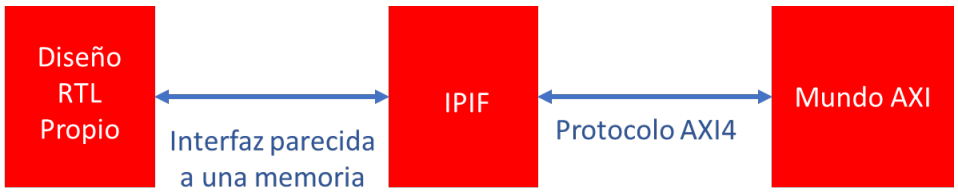


Figura 1-20 Unidades IPIF, esquema básico de conexión

  Con esta solución podemos simplificar el comunicarnos con el AXI, con los tiempos y las señales necesarias, y permite al diseño lógico comunicarse con el IPIF con una interfaz fácil de usar parecida a una memoria, usando direcciones de memoria, write enable, read enable, tamaño de datos a transferir… etc.

- Código autogenerado por Vivado.
  Usando la funcionalidad Package IP de Vivado, se genera un código RTL válido y que tiene la funcionalidad básica de lo que esa interfaz debe tener.
- Usar el Vivado HLS:
  Podemos generar módulos usando código C/C++ y usando las “pragma” de HLS crear una AXI Interfaz para comunicarnos con el módulo que hemos generado.

En este proyecto se ha elegido usar la opción del código autogenerado por Vivado, ya que es una opción muy sencilla y rápida, en la se genera automáticamente el número de registros se desee.

Otra razón por la que elegir el IP Packager, es porque genera un ejemplo de interfaces AXI, en concreto las que se han creado son periféricos con la interfaz AXI4-Lite, también empaqueta todo lo necesario para el funcionamiento del periférico como por ejemplo lugar se puede generar la documentación, los drivers y las constantes, como la dirección base de memoria, que serán usados más tarde por SDK para manejar los periféricos.

La siguiente figura corresponde con un diagrama de bloques de este tipo de solución.

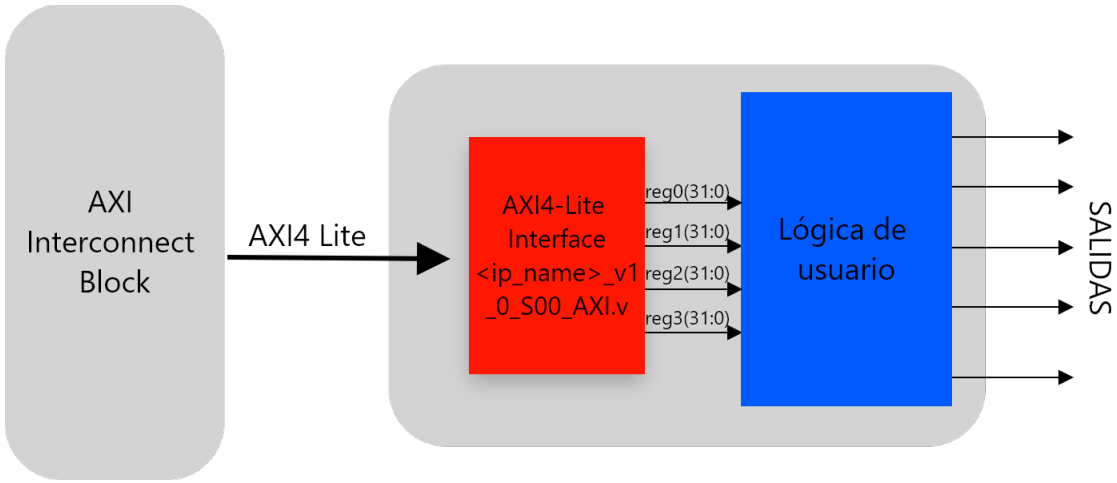


Figura 1-21 Diagrama de bloques de creación de un periférico con AXI4 en IP Packager

Cuando le damos a crear y empaquetar IP en Vivado, y elegimos crear un nuevo bloque AXI4, Vivado nos ofrece un bloque donde están implementadas las comunicaciones. En los bloques que se han creado, se ha elegido usar AXI Lite Interfaces por su menor cantidad de señales y porque además no se requiere el uso de ráfagas (streams).

La CPU usa una de sus interfaces MGP0 o MGP1, para conectarse a la PL. EL MGP se conecta a un AXI Interconnect y este se conectará al bloque por una interfaz AXI Lite.

El intercambio de información entre la interfaz de comunicación y la lógica del driver se realiza mediante registros.

1.9.2.6 Modelado del robot

Este capítulo trata sobre el modelado matemático del robot objeto de estudio, para ello se realizará un esquema simplificado del robot y mediante las ecuaciones de equilibrio estático como dinámico se determinarán las ecuaciones de orden del sistema.

El robot de dos ruedas consiste en un cuerpo largo con dos rudas unidas en un extremo. Para simplificar los cálculos las dos ruedas serán tratadas como una unidad y se asume que el robot viajará en una sola dirección.

Estas son el conjunto de asunciones que se hacen para modelar el robot:

- Las ruedas están siempre en contacto con el suelo, que además experiencia rodadura pura sin deslizamiento.
- Las pérdidas mecánicas y eléctricas se pueden aproximar a cero.
- La respuesta eléctrica es significantemente más rápida que la del sistema mecánico, por tanto, las dinámicas del sistema eléctrico pueden ser obviadas.
- El movimiento del robot está restringido en una sola línea, para que pueda ser analizado como un sistema con movimientos restringidos en dos dimensiones.
- Todas las partes del robot se consideran sólidas rígidas.

- El ángulo de inclinación del péndulo respecto a la vertical es siempre suficientemente pequeño como para permitir la linealización $\sin(\theta) = \theta$.
- La velocidad angular de la inclinación es lo suficientemente pequeña $\dot{\theta}^2 \approx 0$, como para que la fuerza centrífuga sea obviada.

Sea el modelo simplificado el que se muestra en la Figura 1.22, donde actúan dos pesos debido a la rueda ($m_r \cdot g$) y al péndulo ($m_p \cdot g$), donde el par de entrada que generan los motores se transmite directamente a las ruedas (τ_m), siendo F_{fr} la fuerza de rozamiento que se genera entre la rueda y el suelo.

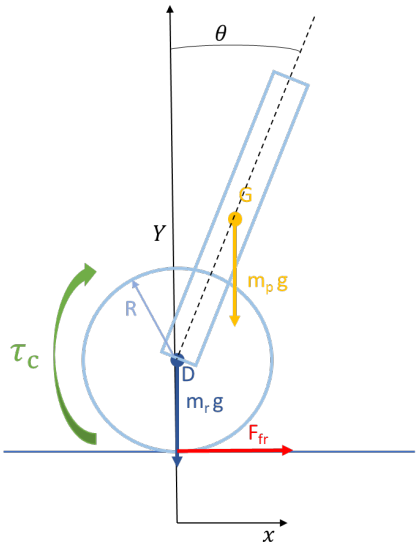


Figura 1-22: Esquema simplificado del robot

Las siguientes magnitudes son las magnitudes de estado, que describen de manera completa el estado del modelo.

- θ el ángulo de inclinación del robot.
- $\ddot{\theta}$ la aceleración angular del robot.
- $\dot{\phi}$ la velocidad de las ruedas.

La siguiente ecuación es la magnitud de control, sobre la que se tiene control y sobre la que se puede actuar para controlar el estado del robot.

- V la tensión de alimentación de los motores y la magnitud de acción de control.

Equilibrio dinámico de la rueda

Entrando en detalle en la rueda y trazando el diagrama de cuerpo libre que se representa en la Figura 1.23, se generarán dos reacciones entre el apoyo de la rueda con el péndulo, N_{xr} y N_{yr}, que en el caso de la barra serán de igual modulo y de dirección contraria (N_{xp} y N_{yp})

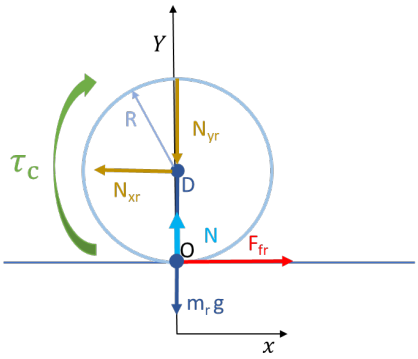


Figura 1-23: Diagrama de cuerpo libre de la rueda

Conocido el modelo y las fuerzas que sobre él actúan se aplican las ecuaciones de equilibrio dinámico en las dos direcciones (X e Y), Ecuación 1.1 y Ecuación 1.2.

$$\sum F_x = m_r \cdot a_r;$$

Ecuación 1-1

$$-N_{xr} + F_{fr} = m_r \cdot a_r\ ;$$

Es necesario escribir las ecuaciones en función de las variables de estado, por tanto, se cambia a_r por $\ddot{x}$.

$$-N_{xr} + F_{fr} = m_r \cdot \ddot{x}_r\ ;$$

Ecuación 1-2

Donde:

- N_{xr} es la fuerza que ejerce el péndulo sobre la rueda.
- F_{fr} es la fuerza de resistencia que se opone al movimiento de la rueda. Es culpable de que la rueda pueda moverse.
- m_r masa de la rueda
- $\ddot{x}_r$ es la aceleración lineal en el eje x.

En el eje y no hay movimiento ni hay variaciones en la velocidad en el eje y, por tanto, no hay aceleración en el eje y.

$$\sum F_y = 0$$

Ecuación 1-3

$$-N_{yr} - m_r \cdot g + N = 0\ ;$$

Ecuación 1-4

Una vez realizado el equilibrio dinámico de las fuerzas, es ahora necesario realizar el cálculo del equilibrio de los momentos. Se usará la ecuación:

$$\sum M_D = I_D \cdot \alpha$$

Ecuación 1-5

En este caso no hay problema con usar la ecuación anterior pues el sumatorio de momentos se hace respecto el centro de masas de la rueda, por lo tanto, no hay que preocuparse por las fuerzas de inercia.

$$-\tau_c + r \cdot F_{fr} = I_r \cdot \ddot{\phi}$$

Ecuación 1-6

Siendo:

- τ_c el torque generado por la caja de cambios.
- r el radio de la rueda.
- F_{fr} la fuerza de resistencia que ofrece el suelo a la rueda.
- I_r el momento de inercia de la rueda.
- $\ddot{\phi}$ la aceleración angular de la rueda.

Equilibrio dinámico del péndulo

Realizando el equilibrio dinámico en el péndulo, Figura 1.24, y de forma similar al procedimiento en el apartado anterior se aplican las ecuaciones de equilibrio dinámico, Ecuación 1.7 y Ecuación 1.9.

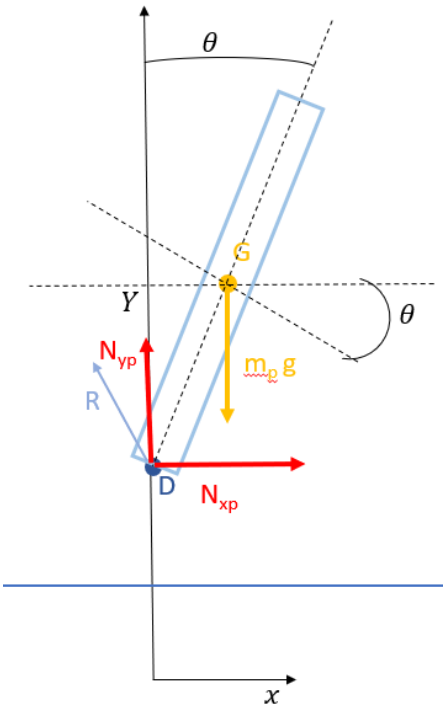


Figura 1-24: Equilibrio dinámico en el péndulo

$$\sum F_x = m_p \cdot a_{px};$$

Ecuación 1-7

$$N_{xp} = m_p \cdot a_{px}$$

Ecuación 1-8

$$\sum F_y = m_p \cdot a_{py}$$

Ecuación 1-9

$$N_{yp} - m_p \cdot g = m_p \cdot a_{py}$$

Ecuación 1-10

Donde la aceleración del péndulo se describe en la siguiente ecuación. Como se puede ver $\overline{a_p}$ depende de la aceleración en la rueda y de la aceleración del péndulo respecto a la rueda. Como no hay desplazamiento relativo, tampoco hay Coriolis.

$$\overline{a_p} = \overline{a_r} + \overline{a_{p-r}} = \ddot{x}\vec{\imath} + \left[-l \cdot \dot{\theta}^2 \cdot \widehat{e_r} + l \cdot \ddot{\theta} \cdot \widehat{e_\theta}\right]$$

Ecuación 1-11

Para el posicionamiento del nuevo sistema de referencias $< \widehat{e_\theta}, \widehat{e_r} >$, Figura 1.25, se tiene en cuenta que el centro de gravedad del péndulo siempre seguirá una circunferencia.

La aceleración que sufre el péndulo en relación con la rueda a_{p-r} siempre tiene una componente tangencial a_t que va en la dirección $\widehat{e_\theta}$, de valor $a_t = l \cdot \alpha = l \cdot \ddot{\theta}$ y una componente normal $a_n$que va en sentido opuesto al vector unitario $\widehat{e_r}$ y de valor $a_n = l \cdot \omega^2 = l \cdot \dot{\theta}^2$.

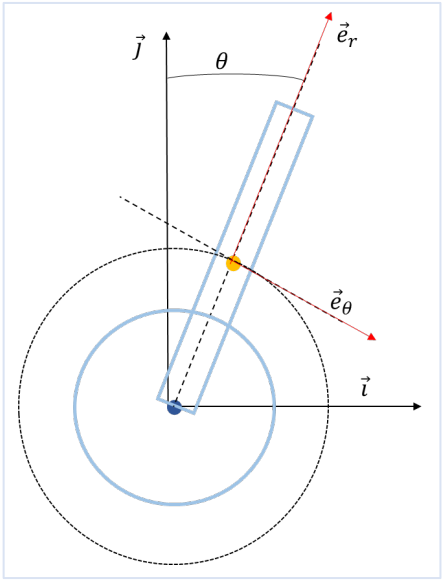


Figura 1-25: Posicionamiento del nuevo sistema de referencia

Los ejes $< \widehat{e_\theta}, \widehat{e_r} >$ tienen una relación con los vectores unitarios $< \hat{\imath}, \hat{\jmath} >$ que se describe con las siguientes ecuaciones:

$$\widehat{e_\theta} = \hat{\imath} \cdot \cos(\theta) - \hat{\jmath} \cdot \sin(\theta)$$

Ecuación 1-12

$$\widehat{e_r} = \hat{\imath} \cdot \sin(\theta) + \hat{\jmath} \cdot \cos(\theta)$$

Ecuación 1-13

Usando las relaciones anteriores en la Ecuación 1.11, obtenemos:

$$\overline{a_p} = \ddot{x}\hat{\imath} + \left[-l \cdot \dot{\theta}^2[\hat{\imath} \cdot \sin(\theta) + \hat{\jmath} \cdot \cos(\theta)] + l \cdot \ddot{\theta}[\hat{\imath} \cdot \cos(\theta) - \hat{\jmath} \cdot \sin(\theta)]\right]$$

Sacando factor común:

$$\overline{a_p} = \hat{\imath}\left[\ddot{x} - l \cdot \theta^2 \cdot \dot{\sin}(\theta) + l \cdot \ddot{\theta} \cdot \cos(\theta)\right] + \hat{\jmath}[-l \cdot \dot{\theta^2} \cdot \cos(\theta) - l \cdot \ddot{\theta} \cdot \sin(\theta)]]$$

Ecuación 1-14

Si se divide este vector en sus componentes y la componente x la sustituimos en la Ecuación 1.8 se obtiene:

$$N_{xp} = m_p \cdot \ddot{x} - m_p \cdot l \cdot \dot{\theta}^2 \cdot \sin(\theta) + m_p \cdot l \cdot \ddot{\theta} \cdot \cos(\theta)$$

Ecuación 1-15

Para el cálculo del sumatorio de momentos se debe tener en cuenta que en la ecuación:

$$\sum M = I \cdot \alpha$$

Ecuación 1-16

Además de tener las fuerzas externas, como la gravedad, se debe tener las fuerzas ficticias, en este caso las de la inercia. Esto se debe a que el sumatorio de momentos no se lleva a cabo en el centro de masas. Estas fuerzas son mejor representadas en la siguiente ecuación, donde $R_{GD} \times m \cdot A_G$ representa la fuerza de la inercia.

$$\sum M_D = I_G \cdot \alpha + R_{GD} \times m_p \cdot A_G$$

Ecuación 1-17

Donde:

- $\sum M_D$ son los momentos calculados desde la unión D.
- I_G es el momento de inercia calculado desde el centro de masas.
- α es la aceleración angular que experiencia el cuerpo
- R_{GD} es el vector que va desde el centro de gravedad G hasta la unión D.
- m_p es la masa del péndulo
- A_G es la aceleración a que se ve sometido el péndulo. Fue calculada en la Ecuación 1.14 con el nombre de $\overline{a_p}$.

Y el esquema que mejor representa los cálculos que se van a llevar a cabo es el siguiente:

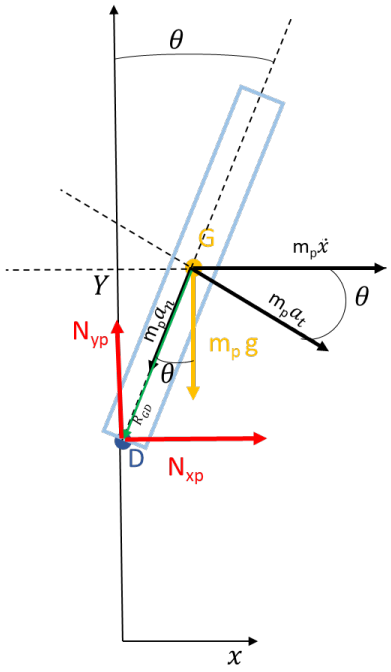


Figura 1-26 Diagrama libre del péndulo 2

Sustituyendo los valores en la Ecuación 1.17 se obtiene:

$$-m_p \cdot g \cdot l \cdot \sin(\theta) = I_p \cdot \ddot{\theta} + R_{GD} \times m_p \cdot \widehat{a_p}$$

Ecuación 1-18

Se debe tener en cuenta que, en el momento de hacer la suma de momentos respecto a D, se considera positivos a aquellos que siguiendo la regla de la mano derecha salen hacia afuera del plano y negativos a los que entran. En el caso de la gravedad, el sentido es hacia adentro, y por tanto, negativo.

$$\overrightarrow{\mathrm{R_{GD}}} = -\mathrm{l} \cdot sin(\theta) \cdot \vec{\imath} - l \cdot cos(\theta) \cdot \vec{\jmath}$$

Ecuación 1-19

$$\overrightarrow{\mathrm{R_{GD}}} \times m_p \cdot \overrightarrow{a_p} = \vec{k}\,[-l \sin(\theta) \cdot m_p \cdot [-l \cdot \dot{\theta}^2 \cdot \cos(\theta) - l\,\ddot{\theta} \sin(\theta)] +$$

$$+ \mathrm{l}\cos(\theta)\, m_p[\ddot{\mathrm{x}} - l\dot{\theta}^2 \sin(\theta) + l\ddot{\theta} \cos(\theta)]]$$

Simplificando la ecuación anterior, se obtiene:

$$\overrightarrow{\mathrm{R_{GD}}} \times m_p \cdot \overrightarrow{a_p} = +l^2 \cdot m_p \cdot \ddot{\theta} + \ddot{x} \cdot l \cdot m_p \cdot \cos(\theta)$$

Ecuación 1-20

Usando esta última ecuación en la Ecuación 1.18 se obtiene el siguiente resultado:

$$\left(I_p + l^2 \cdot m_p\right) \cdot \ddot{\theta} + m_p \cdot g \cdot l \cdot \sin(\theta) = -m_p \cdot l \cdot \ddot{x}\ \cos(\theta)$$

Ecuación 1-21

1.9.2.7 Modelado del motor CC

Para el modelo del motor se tiene en cuenta que está conectado a una reductora de velocidad. Por ello la velocidad $\frac{\dot{\phi}_{motor}}{30} = \dot{\phi}_{rueda}$ y $30 \cdot \tau_{motor} = \tau_{rueda}$. Con esto en consideración se sigue con el modelado del motor.

La ecuación general del circuito de un motor de CC es la siguiente:

$$V(t) = R \cdot i(t) + L \cdot \frac{di}{dt} + e$$

Ecuación 1-22

Siendo:

- $V(t)$ la tensión de alimentación del motor.
- R la resistencia del bobinado del motor.
- L la parte inductiva del bobinado del motor.
- $i(t)$ la corriente que circula por el bobinado.
- e es la fuerza contraelectromotriz que se genera por la rotación del eje del motor.

En un motor eléctrico se sabe además que la fuerza contraelectromotriz, e, es directamente proporcional a la velocidad que gira el eje del motor.

$$e = K_e \cdot \dot{\phi}_m$$

Ecuación 1-23

Donde:

- K_e es una constante
- $\dot{\phi}_m$ es la velocidad a la que gira el eje del motor.

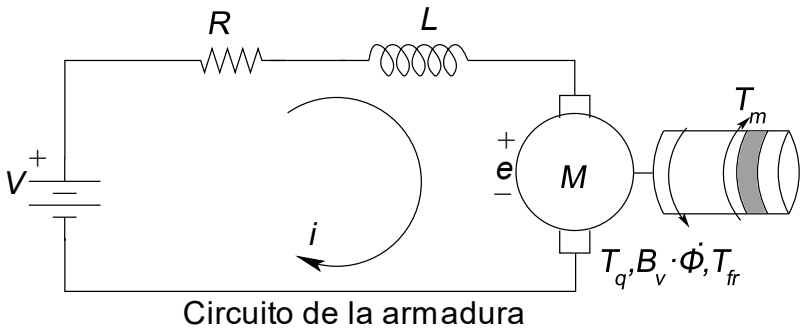


Figura 1-27 Modelo del motor

Aplicando la ecuación de equilibrio dinámico a la carga del motor, es decir a la rueda, se obtiene la siguiente ecuación. Se tiene que tener en cuenta que el sentido de signos escogidos es así porque se quiere expresar que la dirección de giro del eje del motor conectado a una tensión positiva es en sentido contrario a las agujas del reloj:

$$I_c \cdot \ddot{\phi}_c = -\tau_m + B_V \cdot \dot{\phi} + \tau_q + \tau_{fr}$$

Ecuación 1-24

Siendo:

- I_c el momento de inercia de la carga del motor (en este caso será el momento de la rueda respecto al eje, respecto a D en el diagrama de cuerpo libre)
- $\ddot{\phi}_c$ es la aceleración angular que experimenta la carga del motor, en este caso la rueda.
- τ_m el torque que el motor sobre la rueda.
- B_V es el coeficiente de viscosidad del rozamiento lineal.
- $\dot{\phi}$ es la velocidad angular de la rueda.
- τ_q es el torque estático de fricción.
- τ_{fr} es el torque que se opone al movimiento provocado por la fuerza de resistencia (fr). La fr es la que hace que se desplace la rueda.

En el modelo usado para obtener la Ecuación 1-25,el torque estático de fricción τ_q se supone que es igual al torque del motor τ_m mientras $\tau_q < \tau_{q_{max}}$. Esta suposición explicaría la existencia de la zona muerta del motor, la cual es una parte no lineal de modelo.

El valor de τ_m es proporcional a la corriente de armadura, según la siguiente ecuación:

$$\tau_m = K_t \cdot i(t)$$

Ecuación 1-25

Siendo:

- K_t un factor constante.
- $i(t)$ la corriente de la armadura.

Una vez se han obtenido estas ecuaciones se las puede llevar al dominio de las s, utilizando la transformada de Laplace.

Esto quedaría de esta forma:

$$\begin{cases} V(s) = R \cdot I(s) + L \cdot I(s) \cdot s + Ke \cdot \dot{\Phi}(s) \\ I_m \cdot \dot{\Phi}(s) \cdot s = K_t \cdot I(s) - B_v \cdot \dot{\Phi}(s) \end{cases}$$

Ecuación 1-26

Despejando I(s)

$$\begin{cases} V(s) - K_e \cdot \dot{\Phi}(s) = I(s)(R + L \cdot s) \\ I_m \cdot \dot{\Phi}(s) \cdot s + B_v \cdot \dot{\Phi}(s) = K_t \cdot I(s) \end{cases}$$

Ecuación 1-27

Para resolver dividimos las dos ecuaciones y nos queda:

$$\left(V(s) - K_e \cdot \dot{\Phi}(s)\right) K_t = (R + L \cdot s) \cdot \left(I_m \cdot \dot{\Phi}(s) \cdot s + B_v \cdot \dot{\Phi}(s)\right)$$

Ecuación 1-28

$$\dot{\Phi}(s)[(R + L \cdot s) \cdot (I_m \cdot s + B_v) + K_e \cdot K_t] = V(s) \cdot K_t$$

Ecuación 1-29

Finalmente se obtiene la función de transferencia.

$$FT = \frac{\dot{\Phi}(s)}{V(s)} = \frac{K_t}{(I_m \cdot s + B_v) \cdot (R + L \cdot s) + K_e K_t}$$

Ecuación 1-30

A la hora de obtener los valores se tendría una sola ventaja y esta es que los valore de Ke y Kt se pueden considerar iguales si se considera que las pérdidas electromagnéticas se pueden despreciar.

$$P_{mecánica} = P_{eléctrica}$$

Ecuación 1-31

$$\tau_m \cdot \dot{\theta} = e \cdot i(t)$$

Ecuación 1-32

$$K_t \cdot i(t) \cdot \dot{\theta} = K_e \cdot \dot{\theta} \cdot i(t)$$

Ecuación 1-33

$$K_t = K_e$$

Ecuación 1-34

Ahora se quiere conocer el valor de las incógnitas R, k, Bv y Tq. Si se quisiera estimar la función de transferencia de la rueda y como es un sistema estable, basta tener un sistema de adquisición y sistema de alimentación variable.

Se van a llevar a cabo unas simplificaciones de las ecuaciones anteriormente calculadas, para adecuarse a los materiales disponibles en la redacción del proyecto y para facilitar la simulación del modelo.

- Los valores de Bv y Tq no son muy grandes y se pueden despreciar en favor de facilitar la simulación del modelo. Se puede apreciar en la siguiente ecuación, resultado de despreciar Bv y Tq en la Ecuación 1-24, que es la misma que se encontró cuando se hizo el equilibrio dinámico de los momentos de la rueda sobre D:

$$I_m \cdot \ddot{\phi}_m = -\tau_m + \tau_{fr}$$

Ecuación 1-35

- Como ya se ha explicado anteriormente los valores de Ke y Kt se pueden considerar iguales.
- Si se considera la constante eléctrica L/R suficientemente pequeña en comparación con constante mecánica I_m/B_v, se puede simplificar la Ecuación 1-22 despreciando la inductancia L.

$$V(t) = R \cdot i(t) + e$$

Ecuación 1-36

Se va a hacer un resumen de las ecuaciones halladas:

Ecuaciones de la rueda

Nombre	Ecuación	Número
Equilibrio dinámico fuerzas en X.	$-N_{xr} + F_{fr} = m_r \cdot \ddot{x}_r$	1-2
Equilibrio dinámico fuerzas en Y	$-N_{yr} - m_r \cdot g + N = 0$	1-4
Equilibrio dinámico de momentos sobre D	$-\tau_c + r \cdot F_{fr} = I_r \cdot \ddot{\phi}_r$	1-6

Tabla 1-2 Tabla resumen de las ecuaciones de la rueda

Ecuaciones del péndulo

Nombre	Ecuación	Número
Equilibrio dinámico fuerzas en X.	$N_{xp} = m_p \cdot \ddot{x} - m_p \cdot l \cdot \dot{\theta}^2 \cdot \sin(\theta) + m_p \cdot l \cdot \ddot{\theta} \cdot \cos(\theta)$	1-15
Equilibrio dinámico de momentos sobre D	$\left(I_p + l^2 \cdot m_p\right) \cdot \ddot{\theta} + m_p \cdot l \cdot g \cdot \sin(\theta) = -m_p \cdot l \cdot \ddot{x} \cos(\theta)$	1-21

Tabla 1-3 Tabla resumen de las ecuaciones del péndulo

Ecuaciones del motor

Nombre	Ecuación	Número
Fuerza contraelectromotriz	$e = K_e \cdot \dot{\phi}_m$	1-23
Torque que ejerce el motor sobre las ruedas	$\tau_m = K_t \cdot i(t)$	1-25
Equilibrio dinámico de los momentos en el eje motor	$I_c \cdot \ddot{\phi}_c = -\tau_m + \tau_{fr}$	1-35
Ecuación del circuito eléctrico del motor	$V(t) = R \cdot i(t) + e$	1-36

Tabla 1-4 Tabla resumen de las ecuaciones del motor

1.9.2.8 Simplificación de las ecuaciones del modelo

Se van a simplificar las ecuaciones del péndulo, usando las suposiciones del apartado Modelado del robot.

$$\sin(\theta) \approx \theta$$

Ecuación 1-37

$$\cos(\theta) \approx 1$$

Ecuación 1-38

$$\theta^2 \approx 0$$

Ecuación 1-39

Con esto las ecuaciones del péndulo, ecuación 1-15 y ecuación 1-21 quedan:

$$N_{xp} = m_p \cdot \ddot{x} + m_p \cdot l \cdot \ddot{\theta}$$

Ecuación 1-40

$$\left(I_p + l^2 m_p\right) \cdot \ddot{\theta} + m_p \cdot l \cdot g \cdot \theta = -m_p \cdot l \cdot \ddot{x}$$

Ecuación 1-41

La velocidad tangencial es siempre perpendicular a la superficie y la aceleración en el eje Y es siempre 0. Esta es una consecuencia de la consideración de que el modelo solo se puede mover en una línea recta.

La aceleración tangencial de la rueda se corresponde en dirección, sentido y módulo a la aceleración lineal.

Por tanto:

$$a_{t_{rueda}} = a_{lineal_{rueda}} = \ddot{x}$$

Ecuación 1-42

Si se tiene en cuenta que:

$$a_t = r \times \ddot{\phi}$$

Ecuación 1-43

$$\ddot{\phi}_{rueda} = \frac{a_{t_{rueda}}}{r} = \frac{\ddot{x}_{rueda}}{r}$$

Ecuación 1-44

También se debe tener en cuenta, como se mencionó al principio del cálculo de las ecuaciones del motor, debido a que el motor está conectado a una reductora de velocidad se aplican las siguientes ecuaciones $\frac{\dot{\phi}_{motor}}{30} = \dot{\phi}_{rueda}$ y $30 \cdot \tau_{motor} = \tau_{rueda}$.

Una vez dicho esto se hace un cambio en todas las ecuaciones a variables de estado:

De las ecuaciones 1-40, 1-41:

$$N_{xp} = m_p \cdot r \cdot \ddot{\phi}_r + m_p \cdot l \cdot \ddot{\theta}$$

Ecuación 1-45

$$\left(I_p + l^2 m_p\right) \cdot \ddot{\theta} + m_p \cdot l \cdot g \cdot \theta = -m_p \cdot l \cdot r \cdot \ddot{\phi}_r$$

Ecuación 1-46

$$-N_{xr} + F_{fr} = m_r \cdot r \cdot \ddot{\phi}_r$$

Ecuación 1-47

De la Ecuación 1-23 , se aplica que $\dot{\phi}_{motor} = 30 \cdot \dot{\phi}_{rueda}$.

$$e = K_e \cdot \dot{\phi}_m = K_e \cdot 30 \cdot \dot{\phi}_r$$

Ecuación 1-48

A la Ecuacion1-25 se aplica $30 \cdot \tau_{motor} = \tau_{rueda}$

$$\tau_c = 30 \cdot \tau_m = 30 \cdot K_t \cdot i(t)$$

Ecuación 1-49

Si sustituimos esto en las ecuaciones 1-36 y 1-6 respectivamente:

$$V(t) = R \cdot i(t) + 30K\dot{\phi}_r$$

Ecuación 1-50

$$-30 \cdot K_t \cdot i(t) + r \cdot F_{fr} = I_r \cdot \ddot{\Phi}_r$$

Ecuación 1-51

Antes de empezar a juntar las ecuaciones para sacar la descripción en el espacio de estados del robot, se debe tener en cuenta que la fuerza Nxp que es la que ejercen las ruedas sobre el péndulo, se tiene que contar dos veces porque son dos ruedas.

Por tanto:

$$-2 \cdot Nxr = Nxp$$

Ecuación 1-52

Por tanto, la ecuación 1-45 del péndulo:

$$-2N_{xr} = m_p \cdot r \cdot \ddot{\phi}_r + m_p \cdot l \cdot \ddot{\theta}$$

Ecuación 1-53

Haciendo un resumen de las ecuaciones que serán útiles en el cálculo de las funciones de transferencia que definen el sistema:

Nombre	Ecuación	Número
Equilibrio dinámico fuerzas en X.	$-N_{xr} + F_{fr} = m_r \cdot r \cdot \ddot{\phi}_r$	1-47
Equilibrio dinámico de momentos sobre D	$-30 \cdot K \cdot i(t) + r \cdot F_{fr} = I_r \cdot \ddot{\phi}_r$	1-51

Tabla 1-5 Tabla resumen de las ecuaciones de la rueda

Ecuaciones del péndulo

Nombre	Ecuación	Número
Equilibrio dinámico fuerzas en X.	$-2N_{xr} = m_p \cdot r \cdot \ddot{\phi}_r + m_p \cdot l \cdot \ddot{\theta}$	1-53
Equilibrio dinámico de momentos sobre D	$(I_p + l^2 m_p) \cdot \ddot{\theta} + m_p \cdot l \cdot g \cdot \theta = -m_p \cdot l \cdot r \cdot \ddot{\phi}_r$	1-49

Tabla 1-6 Tabla resumen de las ecuaciones del péndulo

Ecuaciones del motor

Nombre	Ecuación	Número
Ecuación del circuito eléctrico del motor	$V(t) = R \cdot i(t) + 30K\dot{\phi}_r$	1-50

Tabla 1-7 Tabla resumen de las ecuaciones del motor

Se unen todas las ecuaciones quedando dos al final de las operaciones, el orden es el siguiente:

- Se despeja i(t) de la ecuación 1-50 y se la sustituye en la ecuación 1-51.
- Se despeja F_{fr} y se lo sustituye en la ecuación 1-47.
- Se despeja N_{xr} y se lo sustituye en la ecuación 1-53.
- A la ecuación 1-49 se la prepara para trabajar con ella.

$$\ddot{\phi}_r\left[\frac{2m_r r^2 - m_p \cdot r^2 - 2I_r}{r}\right] + V\left[\frac{-60K}{r \cdot R}\right] + \dot{\phi}_r\left[\frac{30^2 K^2}{r \cdot R}\right] = \ddot{\theta}\left[m_p l\right]$$

Ecuación 1-54

$$\ddot{\theta}\left[I_p + m_p l^2\right] + \theta\left[m_p l g\right] = \ddot{\phi}_r[-m_p l r]$$

Ecuación 1-55

1.9.2.9 Cálculo de las funciones de transferencia

Pasando las ecuaciones al dominio de Laplace:

$$\dot{\Phi}(s)\left[\frac{s \cdot w_1 \cdot R + 30^2 K^2}{r \cdot R}\right] + V(s)\left[\frac{-60K}{r \cdot R}\right] = \Theta(s)\left[s^2(m_p l)\right]$$

Ecuación 1-56

Siendo:

$$w_1 = 2m_r r^2 - m_p \cdot r^2 - 2I_r$$

Ecuación 1-57

$$\Theta(s)\left[s^2\left(I_p + m_p l^2\right) + m_p l g\right] = \dot{\Phi}(s)[s(-m_p l r)]$$

Ecuación 1-58

Se vuelven a simplificar las ecuaciones para facilitar el cálculo:

$$\dot{\Phi}(s)\left[\frac{s \cdot w_5 + w_6}{r \cdot R}\right] + V(s)\left[\frac{w_7}{r \cdot R}\right] = \Theta(s)[s^2 w_8]$$

Ecuación 1-59

Siendo:

$$w_1 = 2m_r r^2 - m_p \cdot r^2 - 2I_r$$
$$w_5 = w_1 \cdot R$$
$$w_6 = 30^2 K^2$$
$$w_7 = -60K$$
$$w_8 = m_p l$$

$$\Theta(s)[s^2 w_2 + w_3] = \dot{\Phi}(s)[s w_4]$$

Ecuación 1-60

$$w_2 = \left(I_p + m_p l^2\right)$$
$$w_3 = m_p lg$$
$$w_4 = \left(-m_p lr\right)$$

Cálculo de la función de transferencia $\frac{\boldsymbol{\theta(s)}}{\boldsymbol{V(s)}}$

El resultado de despejar $\dot{\Phi}(s)$ en la ecuación 1-60 e insertarla en la ecuación 1-59

$$\frac{\theta(s)}{V(s)} = \frac{-s \cdot w_4 \cdot w_7}{s^3(w_2 w_5 - w_4 w_8 r \cdot R) + s^2 w_2 w_6 + s w_3 w_5 + w_3 w_6}$$

Ecuación 1-61

Siendo:

$$w_1 = 2m_r r^2 - m_p \cdot r^2 - 2I_r$$
$$w_5 = w_1 \cdot R$$
$$w_6 = 30^2 K^2$$
$$w_7 = -60K$$
$$w_8 = m_p l$$
$$w_2 = \left(I_p + m_p l^2\right)$$
$$w_3 = m_p lg$$
$$w_4 = \left(-m_p lr\right)$$

La comprobación con Matlab de los cálculos está en el ANEXO V del documento ANEXOS

Cálculo de la función de transferencia $\frac{\dot{\boldsymbol{\Phi}}(\boldsymbol{s})}{\mathbf{V}(\boldsymbol{s})}$

El resultado de despejar $\Theta(s)$ en la ecuación 1-60 e insertarla en la ecuación 1-59

$$\frac{\dot{\Phi}(s)}{V(s)} = \frac{-s^2(w_2 w_7) - w_3 w_7}{s^3(w_2 w_5 - w_4 w_8 r \cdot R) + s^2 w_2 w_6 + s w_3 w_5 + w_3 w_6}$$

Siendo:

$$w_1 = 2m_r r^2 - m_p \cdot r^2 - 2I_r$$
$$w_5 = w_1 \cdot R$$
$$w_6 = 30^2 K^2$$
$$w_7 = -60K$$
$$w_8 = m_p l$$
$$w_2 = \left(I_p + m_p l^2\right)$$
$$w_3 = m_p lg$$
$$w_4 = \left(-m_p lr\right)$$

1.9.2.10 Cálculo de las constantes del robot

Para poder simular el funcionamiento del robot es necesario conocer las constantes del robot y por ello se ha hecho esta tabla con valores medidos del robot.

Medida	Símbolo	Valor
Resistencia del motor	R	5.2 Ω
Radio de la rueda	r	0.0325 m
Masa rueda	m_r	0.050 Kg
Longitud del péndulo	2·l	0.078 m
Masa del péndulo	m_p	0.66 Kg
Gravedad	g	9,8m/s^2

Tabla 1-8 Constantes del robot

Con estas medidas se puede seguir a calcular:

Momento de inercia de la rueda:

$$I_r = \frac{1}{2} \cdot m_r \cdot r^2 = \frac{1}{2} \cdot 0.05 \cdot 0.0325^2 = 0.0000264\ kg \cdot m^2$$

Ecuación 1-62

Momento de inercia del péndulo:

$$I_p = \frac{1}{2} \cdot m_p \cdot (2 \cdot l)^2 = \frac{1}{2} \cdot 0.66 \cdot 0.078^2 = 0.022\ kg \cdot m^2$$

Ecuación 1-63

Para calcular el valor de K, se va a alimentar al motor a 3 tensiones distintas. A 5V, 8V y 12V.

Esto se hace con el fin de usar la ecuación 1-36, donde si despejamos e:

$$e = V - Ri(t)$$

Al valor de R ya se ha medido con un multímetro. Una vez se tiene e, la fuerza contraelectromotriz, se puede usar la ecuación 1-23 despejando para K.

$$K = e/\dot{\phi}_m$$

Se tiene también en cuenta que la velocidad es la del motor, y la velocidad que da el módulo hardware encoder_motor es la velocidad de la rueda.

$$\dot{\phi}_{rueda} = \frac{\dot{\phi}_{motor}}{30}$$

Los valores de tensión elegidos se han utilizado porque sólo se disponía en el momento de la redacción del proyecto de una fuente de alimentación ATX.

Para realizar el experimento se ha usado el módulo hardware que se ha diseñado para leer los datos de los encoder. Su diseño está explicado en los Anexo II – II del documento Anexos

En Vivado, el esquema hardware que se programará a la FPGA es el siguiente:

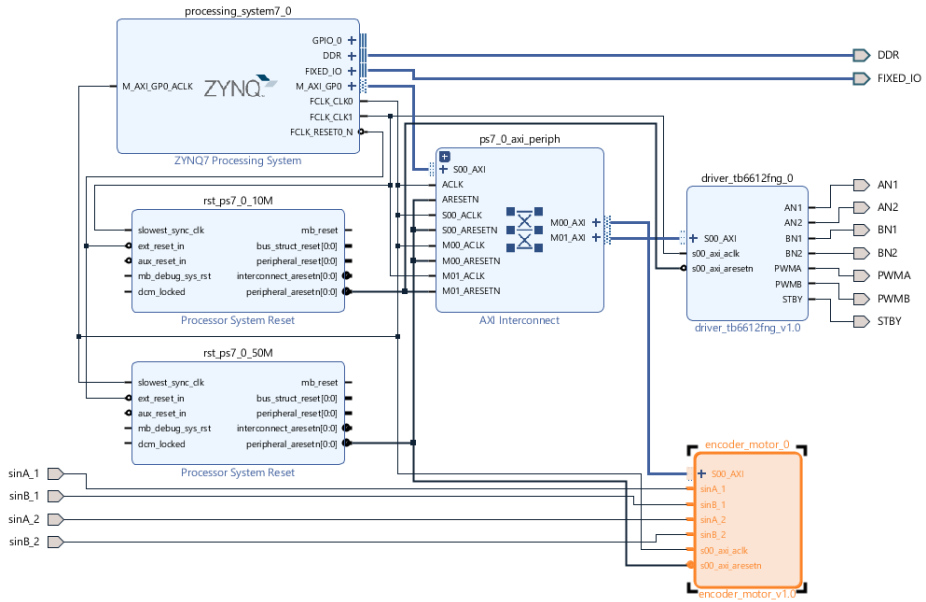


Figura 1-28 Esquema de conexión de los bloques de Vivado

Se va a hacer pruebas con el motor de la derecha, es decir el conectado a las señales sinA_1 y sinB_1.

Se procede a mostrar el código que se ha usado para medir la velocidad de los motores, aunque se explica el funcionamiento completo del módulo en el apartado 1.11.4.

En este código básicamente se ha programado una interrupción del contador privado del CPU. Se ha elegido este porque es más sencillo programarlo.

Cada vez que salta la interrupción, cada 100ms, se lee el número de pulsos en el módulo por AXI-Lite, y se lo divide por el tiempo que tarda en darse la interrupción (100ms). Seguidamente se tiene en cuenta que el motor da 13 pulsos por vuelta y gira 30 vueltas por cada vuelta que gira la rueda.

En la misma interrupción se envía por la UART1 el valor de la velocidad en rpm y rad/s.

Estas son unas imágenes del experimento.

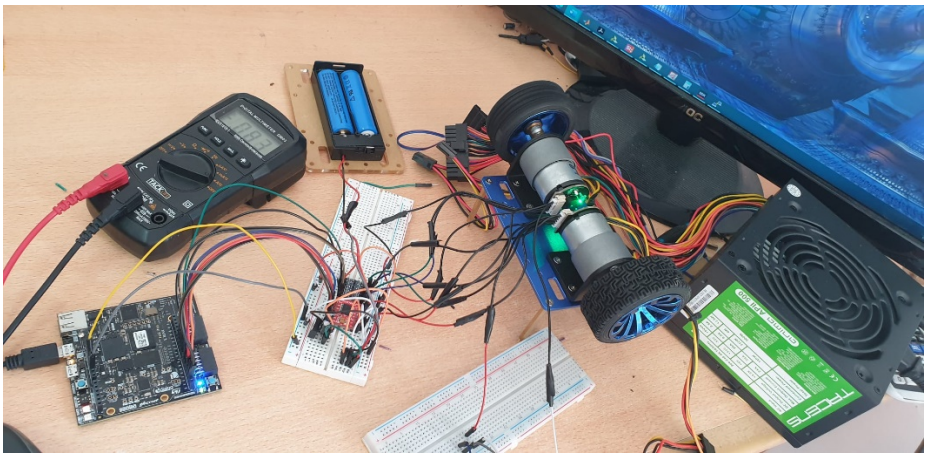


Figura 1-29 Pruebas a 5V

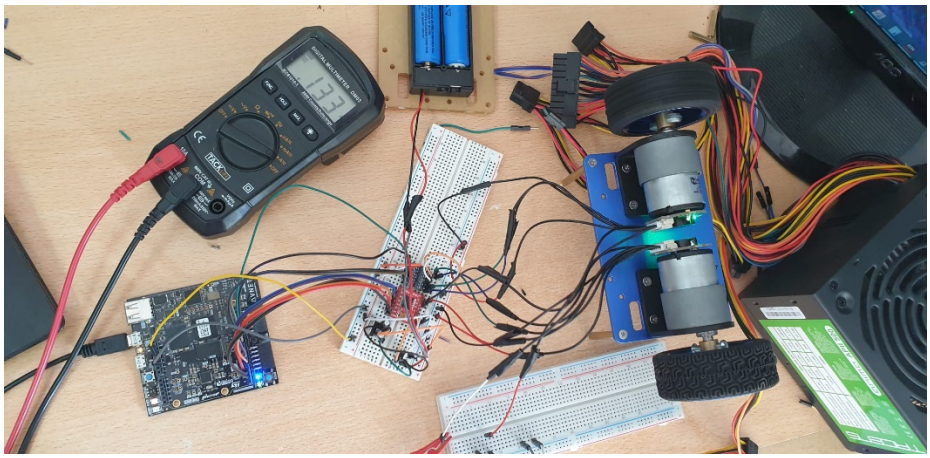


Figura 1-30 Pruebas a 12V

Los resultados obtenidos desde el visor de Putty son:

Siendo alimentado a 5V y midiendo 0.084 A en el multímetro. La velocidad resultante es: 141.53 rpm o 15.03 rad/s

Siendo alimentado a 12V y midiendo 0.133 A en el multímetro. La velocidad resultante es: 336.924 rpm o 35.28 rad/s.

Alimentando el motor a 8V y siendo alimentado por el motor a 0.102 A. La velocidad resultante es: 221.53 rpm o 23.2 rad/s.

Los resultados obtenidos son:

Corriente(A)	**Tensión(V)**	**Velocidad de giro del motor (rad/s)**	**Fuerza contraelectromotriz**	**K**
0.084	5	15.03·30	4.56	0.0101
0.133	12	35.28·30	11.30	0.0106
0.102	8	23.2·30	7.47	0.0107
Valor medio				0.0104

Tabla 1-9 Resultado del experimento

Como se ve la velocidad obtenida desde la consola se multiplica por 30 para obtener la velocidad del motor y respetar las ecuaciones que se han calculado.

1.9.2.11 Simulación del modelo

Una vez obtenido los valores de las constantes se pasa todas las ecuaciones a Matlab y se procede a simular el modelo.

Declaración de constantes

```
R=5.2;%Resistencia eléctrica del motor
K=38;%Constante del motor
r=0.325;%Radio de la rueda
m_p=0.66;%Masa del péndulo
m_r=0.05;%Masa de la rueda
l=0.039;%Mitad de la longitud del péndulo
g=9.8;%Gravedad
I_r=1/2*m_r*r^2;%Momento de inercia de la rueda
I_p=1/2*m_p*(2*l)^2;%Momento de inercia del péndulo
```

Declaración de ecuaciones secundarias

```
w_1=2*m_r*r^2-m_p*r^2-2*I_r;
w_2=I_p+m_p*l^2;
w_3=m_p*l*g;
w_4=-m_p*l*r;
w_5=w_1*R;
w_6=30^2*K^2;
w_7=-60*K;
w_8=m_p*l;
```

Función de transferencia $\frac{\theta(s)}{V(s)}$

```
s=tf('s');
FT1=-s*w_4*w_7/(s^3*(w_2*w_5-w_4*w_8*r*R)+s^2*w_2*w_6+s*w_3*w_5+w_3*w_6);
FT1=minreal(FT1)
```

```
FT1 =

              2.957e04 s
  -------------------------------------
  s^3 - 6.067e06 s^2 + 131 s - 5.082e08

Continuous-time transfer function.
```

Función de transferencia $\frac{\dot{\Phi}(s)}{V(s)}$

```
FT2=(-s^2*w_2*w_7-w_3*w_7)/(s^3*(w_2*w_5-w_4*w_8*r*R)+s^2*w_2*w_6+s*w_3*w_5+w_3*w_6);
FT2=minreal(FT2)
```

```
FT2 =

       -1.064e04 s^2 - 8.915e05
  -------------------------------------
  s^3 - 6.067e06 s^2 + 131 s - 5.082e08

Continuous-time transfer function.
```

1.10 ANÁLISIS DE SOLUCIONES

Antes de llegar a la solución final del diseño para el robot, se han explicado a continuación las distintas opciones que se han tenido en cuenta a la hora de realizar el proyecto. Este apartado recoge las distintas soluciones planteadas en función del componente electrónico o en función de si resolver el problema en software o hardware.

1.10.1 Estudio del esquema final de control

Antes de llevar a cabo el análisis de cada una de las partes del hardware y software, conviene dar una vista de pájaro del sistema.

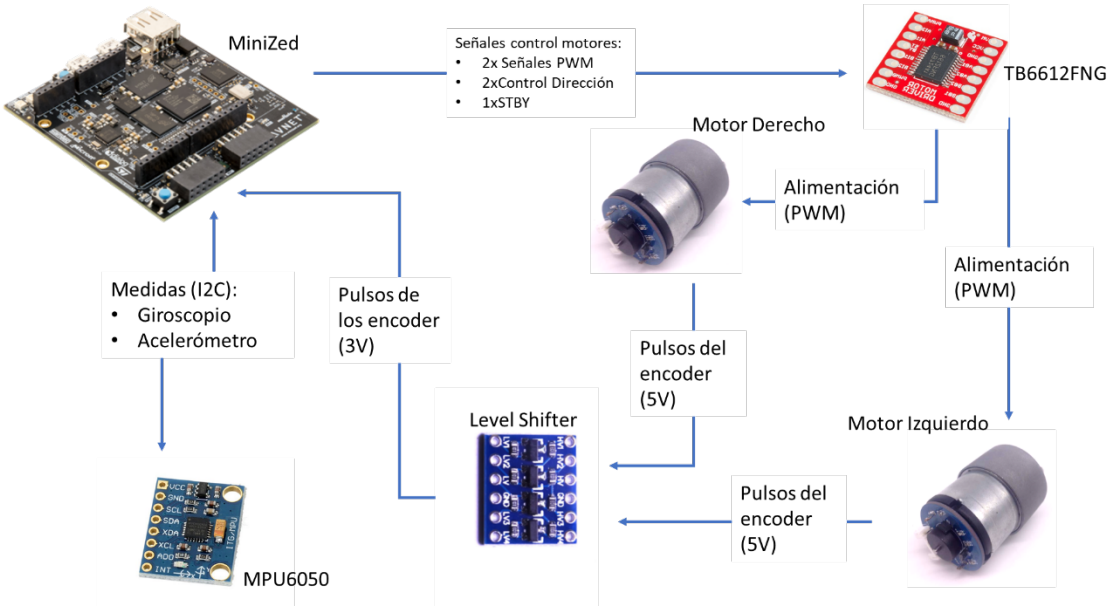


Figura 1-31 Esquema general de señales

En el esquema que se ha presentado, se muestran las señales que estarán presentes en el robot.

Estas señales son:

- Medidas del acelerómetro y giroscopio. Para recibirlas es necesario usar el protocolo I2C.
- Señales de control de los motores. Estas señales van dirigidas a un driver que se encarga de regular la tensión que les llega a los motores.
- Las señales de los sensores hall de los encoder se tienen que pasar por un "level shifter" esto es debido a que la alimentación de ellos debe ser a 5V. Como se ha visto en la creación del módulo de lectura de los encoder, si se alimentan a 3.3V, hay problemas en la medida.

En cuanto a la alimentación de las placas, este es un esquema general del cual se elaborará el esquema en el documento de PLANOS.

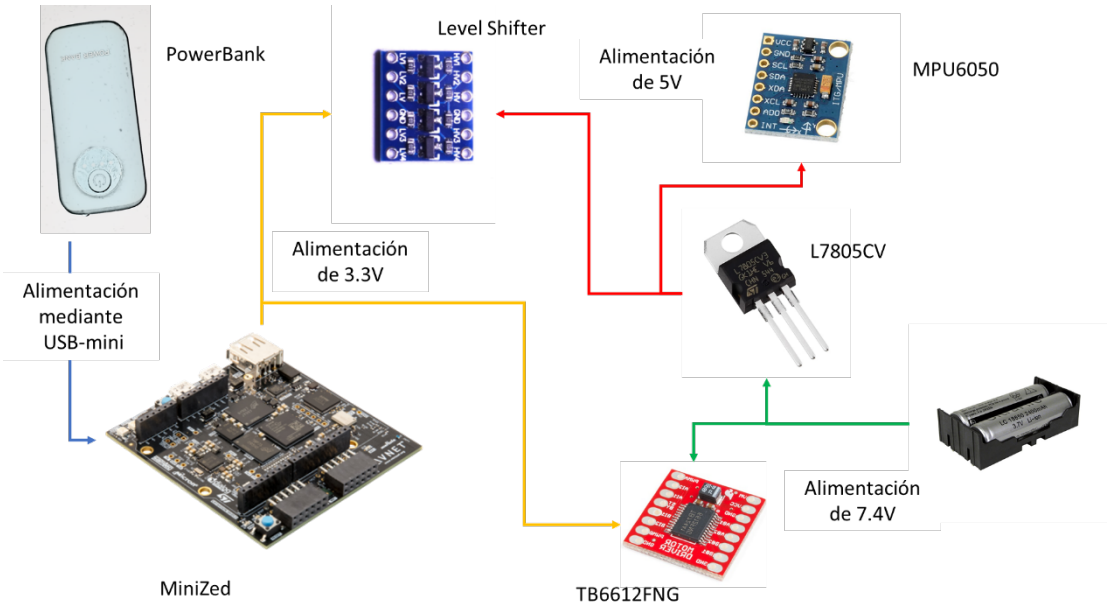


Figura 1-32 Esquema general de alimentación de los módulos electrónicos

Se ha procedido a desarrollar un modelo del robot a escala real en SOLIDWORKS y con ello tener claro las dimensiones del robot y con ello fijar una de las condiciones para realizar el modelo del robot.

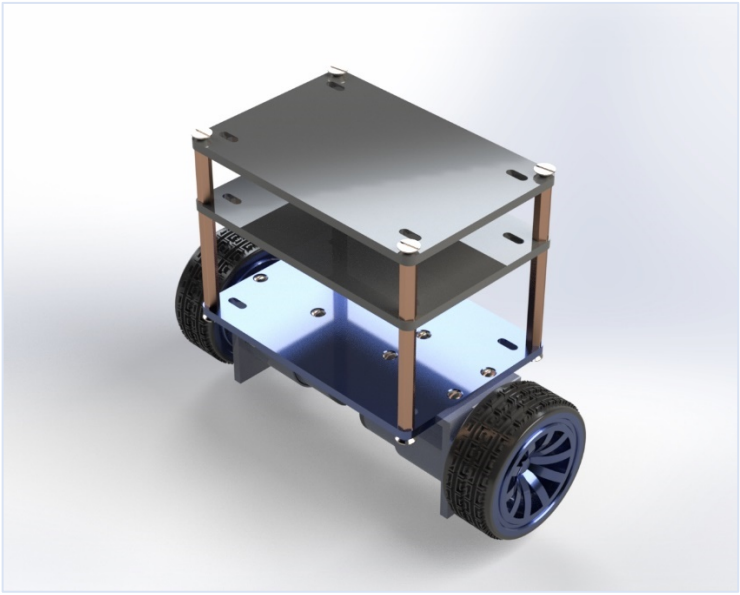


Figura 1-33 Modelo del robot en 3D

Una vez realizado una vista muy por encima de las partes del robot, se va a proceder a explicar cada una de ellas, y por qué se las ha escogido.

1.10.2 Placa de desarrollo

Desde el comienzo del proyecto se ha escogido la utilización de lógica programable, dadas las prestaciones de esta tecnología y el interés propio en el desarrollo de sistemas basados en estos dispositivos.

Existen 3 opciones principales en esta elección:

- Usar un dispositivo soft-core programado dentro de una FPGA, como MicroBlaze, para que realice el control del robot y las comunicaciones, permitiendo también el desarrollo de periféricos en la lógica programable.
- Usar un dispositivo hard-core, un CPU físico, el cual iría unido a una FPGA. En esta ruta la mejor opción es un dispositivo ZYNQ-7000 SoC, el cual tiene en su interior dos partes. Una parte de software programable (PS) y otra parte de lógica programable (PL). La gran ventaja de una ZYNQ respecto a una solución de dispositivos únicos es la gran conectividad que ofrece con los puertos AXI que se encuentran entre sus dos partes. La parte del control del robot y comunicaciones con los dispositivos externos por I2C se realizaría en el CPU, ya que es mucho más sencillo programar en este dispositivo secuencial esas tareas. Siempre quedando la opción de crear periféricos hardware específicos para distintos módulos.
- La tercera opción es desarrollar tanto el control del robot, como la obtención de los valores del acelerómetro y giroscopio mediante I2C, como el desarrollo de los periféricos para controlar los encoder y el driver de los motores, todo en hardware.

Se analizan las opciones dependiendo de:

- La velocidad con que se pueda llevar a cabo la solución.
- Capacidad de reutilizar el hardware programable.
- Opciones de placas de desarrollo en las que se disponga de dispositivos de comunicación inalámbrica.
- Precio de la placa de desarrollo.
- Disponibilidad de la documentación y de material educativo para el dispositivo.

Finalmente se ha escogido la opción de usar un dispositivo ZYNQ. Las razones son las siguientes:

- Disponibilidad en un solo IC de un CPU muy potente y de una FPGA no muy pequeña.
- Gran capacidad de comunicación entre el CPU y la FPGA, con una interfaz con muy poca latencia. La interfaz sigue el protocolo AXI4.
- Gran capacidad de reutilización del código programado, ya que los “dos grandes” Xilinx e Intel han puesto como estándar el protocolo AXI para interactuar entre los bloques IP disponibles. Además, hay disponibles comercialmente gran cantidad de IP cores que venden distintas empresas para acelerar el proceso de diseño.
- Familiaridad con Vivado y aunque nunca se había programado con IP INTEGRATOR, sí se había programado a lo largo de la carrera a nivel RTL. Por eso, se ha escogido usar una opción SoC de Xilinx respecto a las que ofrece Intel.

Como placa de desarrollo se ha escogido una basada en Single-Core ZYNQ 7007s, la MiniZed. Esta placa ofrece un módulo de conexión inalámbrica Bluetooth y WiFi. Además, ofrece unos pines compatibles Arduino aparte de los PMOD, lo cual puede permitir por la popularidad de los Arduinos, conectarle directamente un SHIELD.

En el paquete la MiniZed se incluía una licencia de desarrollo SoC que durante un año permite obtener actualizaciones de los productos de software de Xilinx.

Se ha encontrado material educativo desde el proveedor de la placa AVNET en [5] y por parte de un investigador en [6]. También se ha encontrado mucha ayuda en los foros de Xilinx.

1.10.3 Driver de los motores

Es necesario escoger un dispositivo de potencia que permita a la placa controlar los motores. Debido a la gran corriente que requieren estos dispositivos, si los conectáramos a las salidas de los pines, quemaríamos la placa.

Por ello se requiere de un dispositivo que:

- Sea capaz de manejar el motor descrito en el ANEXO IV - I del documento ANEXOS.
- Tenga documentación disponible y permita una fácil comprensión de su funcionamiento.
- Sus conexiones sean sencillas y el control de los motores esté contenido en un solo IC.
- Proteja a la placa de corriente inversa.

Las opciones de drivers de motores son varias, entre ellos se destacan: L293D y el TB6612FNG.

Finalmente se ha escogido el módulo TB6612FNG, descrito en el ANEXO IV - III del documento ANEXOS.

Este dispositivo permite el control de los dos motores en un solo IC, con un voltaje máximo de 15V y una corriente media de 1.2 A por fase sobrepasa con un margen suficiente los requerimientos del motor.

La documentación sobre este dispositivo se ha encontrado en Sparkfun en [7].

1.10.4 Baterías

Se debe alimentar al circuito, a la placa y a los motores. El tiempo en que puede estar circulando el robot depende de la elección que hagamos de qué batería usar.

Los requerimientos son:

- Capacidad para generar toda la corriente necesaria para alimentar tanto a los motores como al circuito eléctrico de control.
- Es necesario que dure al menos diez minutos la batería.
- Sea recargable.
- Pese poco.

Las opciones de batería son varias, así como la tecnología implicada en crearlas:

- Baterías plomo ácido: Ofrecen la mayor seguridad en cuanto a cortocircuitos y sobrecargas, pues no van a incendiarse ni explotar. El gran inconveniente que tienen son su gran peso en relación con la carga que soportan.
- Baterías LiPo: Ofrecen la mayor corriente máxima en funcionamiento normal, por ello son muy usadas en drones y coches RC que requieren mucha corriente. Son una opción muy interesante, pero son bastante peligrosas en cuanto a que si se cargan mal (no se regula bien la tensión entre celdas) o si se sobrecargan pueden quemarse.

- Baterías Li-ion: Son baterías usadas en diversas industrias, tanto en patinetes eléctricos como en portátiles. Es de destacar que pueden contener la mayor carga por unidad de peso con respecto a las opciones antes citadas. Son ligeras y se ofrecen formas de cargarlas que son bastante baratas.

Finalmente se ha escogido usar baterías Li-ion en concreto dos baterías Samsung 18650 3450mAh de 3.7V cada una. Han sido escogidas especialmente porque ofrecen una opción muy compacta y ligera. Serán puestas en serie y serán sujetadas al robot por una carcasa.

1.10.5 Regulador de tensión

La lógica de los dispositivos debe ser regulada, ya que la que ofrecen las baterías ofrecen 7,4V como tensión nominal, y algunos dispositivos se romperían si los conectamos a este voltaje.

Para ello es necesario regular la tensión de las baterías a 5V para alimentar a los dispositivos.

Lo que se requiere del dispositivo es lo siguiente:

- Que regule de 7.4V a 5V sin sobrecalentarse.
- Que permita el paso de bastante corriente, al menos 1 A, para poder alimentar a todos los dispositivos.

Finalmente se ha escogido el regulador lineal L7805CV, ya que puede dejar pasar hasta 1.5 A, por tanto, puede alimentar a todos los dispositivos de control. Una imagen del dispositivo y de sus pines con un esquema de conexión general se encuentra en el ANEXO IV - V del documento ANEXOS.

1.10.6 Sensor de orientación

Es necesario un sensor del que se pueda obtener información sobre la orientación del robot.

Los requerimientos son:

- El sensor debe ofrecer suficientes datos como para realizar una medida de la orientación que sea fiable.
- El sensor debe ofrecer una forma de comunicarse con el CPU.
- Tiene que estar contenido en un módulo con tensiones compatibles con la placa de desarrollo (3.3V).

Las opciones que se cuentan son:

- El sensor LIS2DS12, que es un acelerómetro con un sensor de temperatura. Este sensor está incluido en la placa de desarrollo y permite obtener los datos sin necesidad de realizar conexiones externas.
- El módulo MPU 6050, que es un acelerómetro con un giroscopio. La gran ventaja de este sensor es que el giroscopio y el acelerómetro se complementan muy bien cuando se hace sensor fusión. Por ejemplo, el acelerómetro ofrece buenas lecturas a corto plazo. Este sensor ofrece como forma de comunicación I2C.

A la hora de elegir el módulo se ha tenido en cuenta que los acelerómetros aparte de medir la dirección de la gravedad de la tierra, también miden las aceleraciones a que es sometido el sistema.

Por ejemplo, cuando el acelerómetro se encuentra en el extremo de un cuerpo que gira o cuando tenemos una aceleración lineal. Por ello es necesario corregir para aceleraciones lineales, como se puede ver en el esquema siguiente si tenemos el modelo del sistema podemos predecir las aceleraciones lineales y eliminarlas de las medidas.

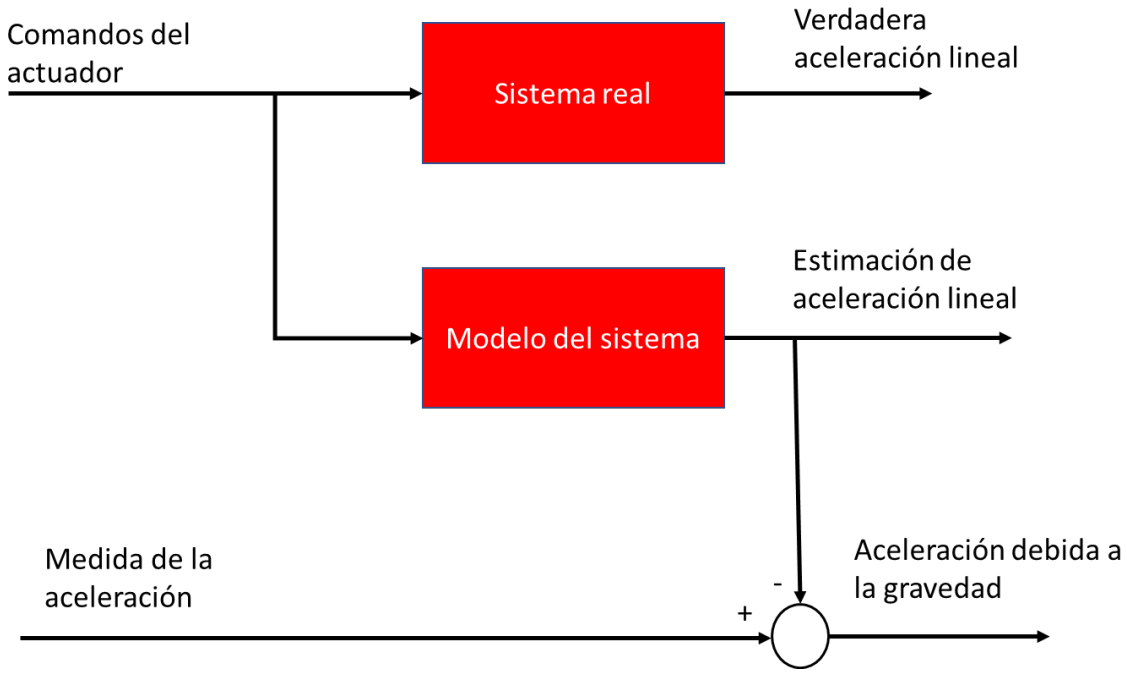


Figura 1-34 Esquema de eliminación de aceleraciones lineales

Esto se puede llevar a cabo usando el modelo del sistema y usando los comandos a los actuadores para predecir la aceleración lineal

Si no podemos predecir la aceleración lineal o tenemos unas perturbaciones externas que son muy grandes, se pueden ignorar aquellas medidas en que la magnitud de la aceleración medida sobrepase la de la gravedad en un cierto margen.

Pero si añadimos un giroscopio se pueden mejorar las medidas. El giroscopio nos da la velocidad angular en los ejes del cuerpo. Si multiplicamos esta velocidad angular por el tiempo de muestreo, obtenemos los grados que ha girado el objeto durante ese tiempo. Si conocemos la orientación del objeto en el anterior periodo de muestreo podemos añadirlo a la estimación actual y obtener la orientación actual.

Si el objeto no está girando, la medida de $\Delta\theta=0$ y por tanto la orientación no variaría. A este proceso se repetiría a cada tiempo de muestreo, podemos saber la orientación del objeto al cabo del tiempo.

El siguiente esquema representa muy bien lo que acabo de explicar:

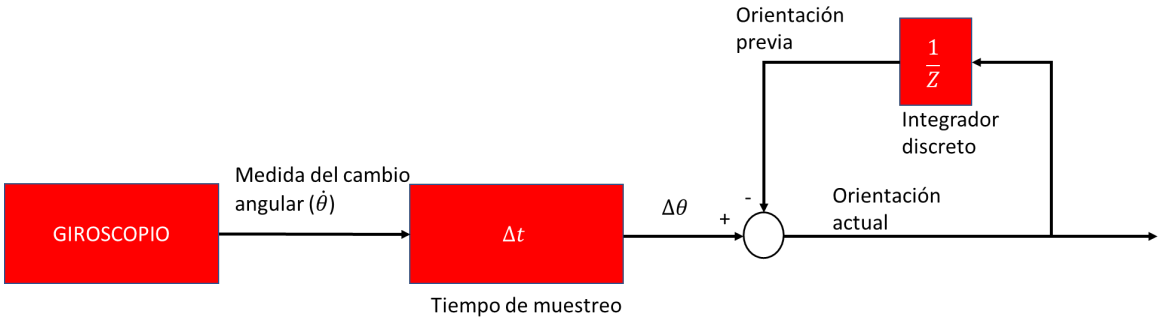


Figura 1-35 Esquema de funcionamiento del giroscopio

Se tiene los siguientes inconvenientes:

- Se necesita conocer la orientación inicial.
- Los sensores no son perfectos, tienen bias (tendencias a dar un valor constante aun cuando debería dar cero) y otros problemas con ruidos de alta frecuencia.

El problema que se tiene es que, al hacer la integración para hallar la variación del ángulo, hace que el ruido y bias de los sensores se vaya añadiendo con el tiempo y ocurra lo que se conoce como drift.

Resumiendo, se puede obtener la orientación del objeto de dos formas, pero cada una tiene sus ventajas y desventajas.

Acelerómetro:

- Produce medidas absolutas.
- Sus medidas se ven corrompidas por disturbaciones comunes.

Giroscopio:

- Produce una medida relativa.
- Necesita la orientación inicial.
- Sufre de “drift” a lo largo del tiempo.

Pero se puede combinar la medida de los sensores mediante los siguientes filtros:

- Complementario
- Kalman
- Madgwick
- Mahony

Estos filtros siguen fundamentalmente los mismos pasos:

- Inicializan la orientación, manualmente o usando los valores iniciales del acelerómetro.
- Mas tarde, usan la gravedad para corregir el drift del giroscopio.

La ventaja principal es que las medidas del giroscopio y el acelerómetro se complementan entre sí y con los filtros se obtiene una medida estable a lo largo del tiempo.

Por ello se escoge el módulo MPU 6050 que incluye un giroscopio y un acelerómetro. Además, se puede interactuar con él de forma fácil con el protocolo de comunicación I2C.

1.11 RESULTADOS FINALES

1.11.1 MiniZed

Para el desarrollo de la solución se ha escogido la placa MiniZed como placa de desarrollo.

MiniZed es una placa de desarrollo de la Single-Core ZYNQ 7007s. Esta placa está orientada al desarrollo de prototipado aplicaciones de bajo coste.

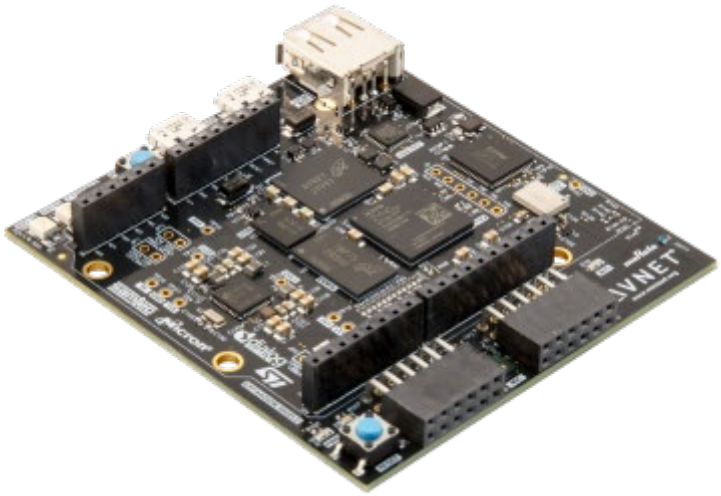


Figura 1-36 MiniZed

Esta placa ofrece una ventaja clara en cuanto a un Arduino UNO. Las principales ventajas son conectividad, CPU mucho más rápido, mayor y más rápida memoria (DDR3L funcionando a 666.66 MHz) y capacidad de diseñar aceleradores y periféricos hardware customizados.

Tiene un módulo para conectividad WIFI y Bluetooth dentro de la misma placa a través del módulo inalámbrico Murata “Type 1DX”.

El procesador es un Single-Core ARM CORTEX-A9 MP Core de 32bits con una velocidad de hasta 667MHz. Esto se une a su extensión NEON, que le permite aumentar en mucho la velocidad de procesamiento, sobre todo de las operaciones SIMD. También incluye una Floating Point Unit de simple o doble precisión.

Esto se une a una FPGA con espacio suficiente como para desarrollar varios aceleradores o periféricos que puede ir a una velocidad de CLK de 250 MHz, creando una combinación muy poderosa.

La placa de desarrollo además tiene unos conectores ARDUINO que resultan muy familiares y facilitan la conexión final del proyecto.

1.11.2 Control sobre la velocidad de las ruedas, módulo hardware AXI-Lite driver_tb6612

Para el control de las ruedas se ha escogido como driver el módulo TB6612, ya que ofrece el control de ambos motores en un solo IC. Los pines de este módulo y sus funciones están documentados en el ANEXO IV - III del documento ANEXOS.

Para el control de este IC se ha diseñado un periférico en el PL, que se ha llamado AXI-Lite driver_tb6612. Este periférico ha sido diseñado y simulado desde su parte lógica, como a la hora de comunicarse con el CPU. Este desarrollo se ha llevado a cabo en el *ANEXO I - II* del documento ANEXOS.

Cuando se conecte el módulo al bloque ZYNQ7 Processing System, debe conectarse a una fuente de CLK de 10MHz.

Si se quiere programar el módulo desde el CPU es necesario escribir a las direcciones de memoria adecuadas, para facilitar esto, SDK autogenera la definición de su dirección base en el valor XPAR_DRIVER_TB6612FNG_0_S00_AXI_BASEADDR, del archivo “xparameters.h”.

La librería que se ha diseñado para trabajar con este módulo “driver_tb6612fngg.h” tiene todo su código en el *ANEXO I - I* del documento ANEXOS.

Si se quiere usar las funciones, como driver_motor_en() que habilita o deshabilita el valor del registro “en” en el módulo, se debe inicializar la variable driver_add de la siguiente manera:

u32 driver_add= XPAR_DRIVER_TB6612FNG_0_S00_AXI_BASEADDR;

Esto se debe hacer en el archivo en que se vayan a usar las funciones de la librería “driver_tb6612fngg.h”, ya que todas las funciones usan ese valor para dirigirse a la dirección base del periférico.

La siguiente función debe ser usada antes de intentar modificar las señales PWM del módulo.

Esta función se encarga de activar el enable del bloque lógico y con ello activar su funcionamiento.

void driver_motor_en(u32 num) ;

Para controlar la velocidad de los motores se genera una señal PWM cuyos valores pueden variar de 0 a 255.

Para controlar la velocidad del motor derecho se debe escribir en la siguiente función:

void driver_motor_timA(u32 num);

Para controlar la velocidad del motor izquierdo se debe escribir en la siguiente función:

void driver_motor_timB(u32 num);

El módulo también dispone de la opción de cambiar el sentido de giro de los motores:

Donde el primer argumento de la función es la dirección del motor derecho, correspondiendo el 0 con la dirección de giro en sentido contrario a las agujas del reloj.

void driver_motor_dir(u32 dirA,u32 dirB);

1.11.3 Adquisición de las señales de los encoder

Para adquirir la información de los encoder, se ha diseñado un módulo hardware que se ha diseñado para que una vez activado (escribiendo un 1 en el Offset 0 y 16) esté continuamente atento a las señales de las ruedas. Su diseño y simulación, así como la información de sus registros están en el ANEXO II del documento ANEXOS.

Hay diversas funciones desarrolladas en C con las cuales se puede acceder a los registros y que están comentadas en el ANEXO II - I del documento ANEXOS.

Primero se debe instanciar la dirección base de la memoria, para que las funciones puedan ser utilizadas. Esta instanciación se hará en la variable externa dirEncoder.

```
u32 dirEncoder=XPAR_ENCODER_MOTOR_0_S00_AXI_BASEADDR;
```

Entre las funciones que están disponibles para manejar el periférico se puede destacar:

La función que permite activar (1), desactivar el módulo(0) o reiniciar los valores de los pulsos(2).

Esta función es para el módulo encoder_logic1:

```
void write_encoder_motor_en_1(u32 value);
```

Esta función es para el módulo encoder_logic2:

```
void write_encoder_motor_en_2(u32 value);
```

La siguiente función permite calcular la velocidad de la rueda en una interrupción.

Para la rueda derecha:

```
void vel_rueda1(float time, float* rpm, float*rads);
```

Para la rueda izquierda:

```
void vel_rueda2(float time, float* rpm, float*rads);
```

Se tiene en cuenta que el valor de time, es el tiempo en que tarda en suceder la interrupción.

1.11.4 Adquisición de los valores del MPU6050

Para la adquisición de los valores del MPU6050 se ha tenido que crear dos funciones para leer y escribir por I2C.

Además, se ha tenido que declarar los offset de los registros que se necesitan para el funcionamiento.

Cabe destacar que para inicializar el módulo y que el CLK de referencia del GYRO_X sea usado por el módulo, siempre se tiene que escribir un 1 en el registro PWR_MGMT_1 con offset 0x6B.

Con el registro RW_SMPLRT_DIV con offset 0x19 se modifica la frecuencia con que genera las muestras el Giroscopio. En la función de inicialización se ha escogido 1KHz y por tanto se ha escrito un 7 en este registro.

Una vez finalizado esto el módulo está funcionando y uno ya es capaz de leer los registros de las medidas del giroscopio y del acelerómetro.

Pero se debe tener en cuenta que las aceleraciones y la velocidad con que gira el módulo puede ser un poco rápida para las escalas por defecto.

Aunque se reduce la sensibilidad se ha creído conveniente cambiar los valores de las escalas a 4g y a 2000 rad/s.

Para ello se escribe en los registros ACCEL_CONFIG con dirección 0x1C y GYRO_CONFIG con 0x1B con los valores 0x08 y 0x18 respectivamente.

Posteriormente si se realizan las medidas, se puede observar que el giroscopio tiene unos valores aun cuando está quieto. Para corregirlo se realiza una calibración cuando se inicializa el módulo que genera unos valores para unas variables globales que restarán a todos los datos medidos por el giroscopio.

Si se quiere que el módulo de una señal de interrupción a la placa cada vez que termine un ciclo de medida se debe escribir un 0x01 al registro INT_ENABLE y si se quiere que se limpie esta interrupción cada vez que se realice una lectura se escribe un 0x10 al registro INT_PIN_CFG.

El código de este módulo está en el ANEXO IV - VI del documento ANEXOS.

Una vez obtenidas las medidas de los sensores es necesario fusionar sus valores para obtener los valores de inclinación. Para esto se ha usado un filtro complementario.

1.11.5 Generación de interrupciones por el SCU Private Timer

Las interrupciones pueden ser generadas por distintos módulos en el PS e incluso por módulos hardware. La mejor opción para una interrupción periódica es el uso de un contador, al cual se le dará un valor el cual irá descontando uno a uno hasta que llegue a cero y genere la interrupción.

En la siguiente figura se muestran los contadores que existen a nivel de sistema:

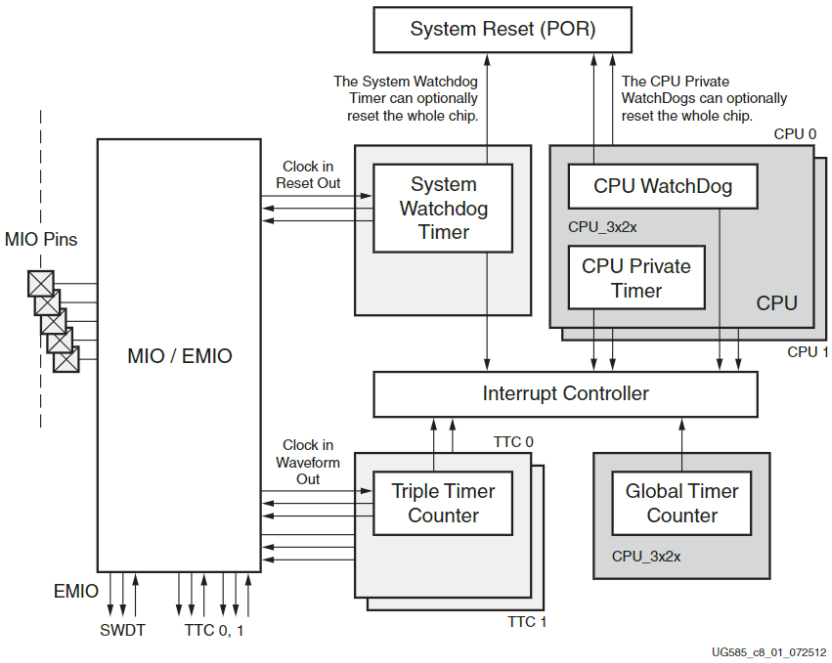


Figura 1-37 Timers

A la hora de la implementación se ha escogido el SCU Private Timer, un contador que pertenece al CPU, por tanto, no es necesario programarlo en Vivado y se puede activar escribiendo en los registros adecuados.

Para manejarlo a nivel alto se han usado las librerías de Xilinx, pero aun así ha sido necesario generar unas funciones por encima que se encarguen de programar la interrupción.

Para manejar las funciones es necesario declarar las siguientes constantes:

Identificación del módulo SCU Private Timer

```
#define TIMER_DEVICE_ID          XPAR_XSCUTIMER_0_DEVICE_ID
```

Generic Interrupt Controller (GIC) Identificación 0

```
#define INTC_DEVICE_ID           XPAR_SCUGIC_SINGLE_DEVICE_ID
```

Numero de identificación de la interrupción

```
#define TIMER_IRPT_INTR          XPAR_SCUTIMER_INTR
```

Declaración de las estructuras que contendrán los valores importantes de las funciones, desde su dirección base hasta su estado actual. Esta definición de estas estructuras debe hacerse dentro de la función principal.

Instancia del SCU Private Timer:

```
XScuTimer TimerInstance;
```

Instancia del GIC

```
XScuGic IntcInstance;
```

Una vez realizado esto, se puede empezar a usar las funciones que se han programado para inicializar los valores de las instancias del SCU Private Timer y del GIC y para linkear el handler de la interrupción del timer con la función Handler Timer. En esta función se programa lo que tiene que hacer el CPU cuando sucede la interrupción.

void HandlerTimer(void *CallBackRef)

La primera función que se debe usar es para Inicializar la instancia del SCU Private Timer.

InicializaTimer(TIMER_DEVICE_ID,&TimerInstance);

La siguiente función habilita la interrupción del SCU Private Timer y hace el link del handler de la interrupción a la función Handler Timer.

HabilitaSCUTimer(&IntcInstance,&TimerInstance,TIMER_IRPT_INTR);

Las siguientes funciones programan al SCU Private Timer como contador descendente, que una vez encendido se vuelve a cargar de manera automática y cargado con el valor TIMER_LOAD_VALUE.

XScuTimer_EnableAutoReload(&TimerInstance);

XScuTimer_LoadTimer(&TimerInstance, TIMER_LOAD_VALUE);

XScuTimer_Start(&TimerInstance);

El código de las funciones está en ANEXO VI - I del documento ANEXOS.

1.11.6 Funcionamiento del controlador PID en el tiempo discreto

Para el equilibrio del robot se ha usado un controlador en el tiempo discreto. El siguiente esquema refleja cómo se realizará el control para equilibrar el robot. Como cuando el péndulo está en equilibrio la inclinación que mide es 0, se ha usado ese valor como referencia del lazo de control.

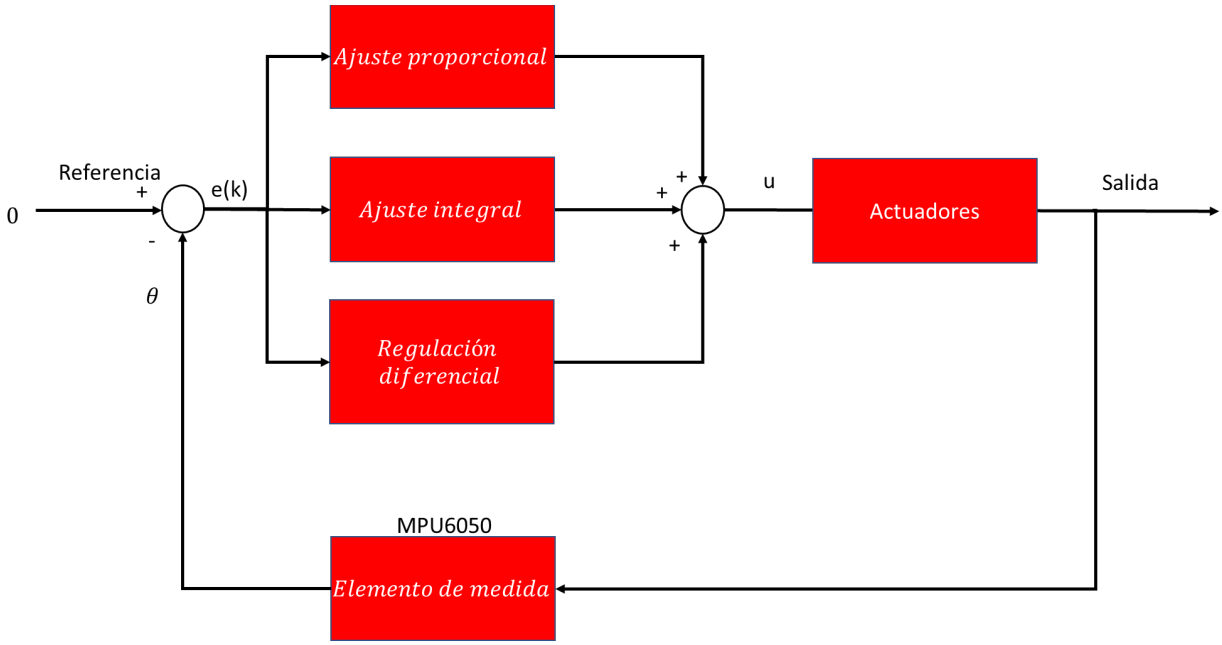


Figura 1-38 Esquema de control

En cada bloque del PID se ha realizado la siguiente programación en el CPU para hallar la acción de control del controlador.

$$Ajuste\ proporcional = kp \cdot e(k)$$
$$Ajuste\ integral = ki \cdot (e_{integral} + e(k) \cdot T_{muestreo})$$

$$Regulación\ diferencial = kd \cdot \big(e(k) - e(k-1)\big)/T_{muestreo}$$

Siendo:

- $e(k) = Referenica - inclinación\ actual$
- $e_{integral}$ es la integral del error e(k)
- $T_{muestreo}$ es el tiempo que tarda en actualizarse el error, tiempo de muestreo, en el caso del robot se ha escogido uno de 10ms

Para hallar los coeficientes de kp, ki y kd ha sido necesario modificarlos hasta que el robot responde de manera adecuada y se equilibra por sí sólo.

Hay una limitación importante en este tipo de control, y es que no se controla la velocidad y por ello si el robot se encuentra en una pendiente, éste tenderá a acelerar cuesta abajo.

1.11.7 Programa final

Aquí se muestra un esquema del funcionamiento del programa.

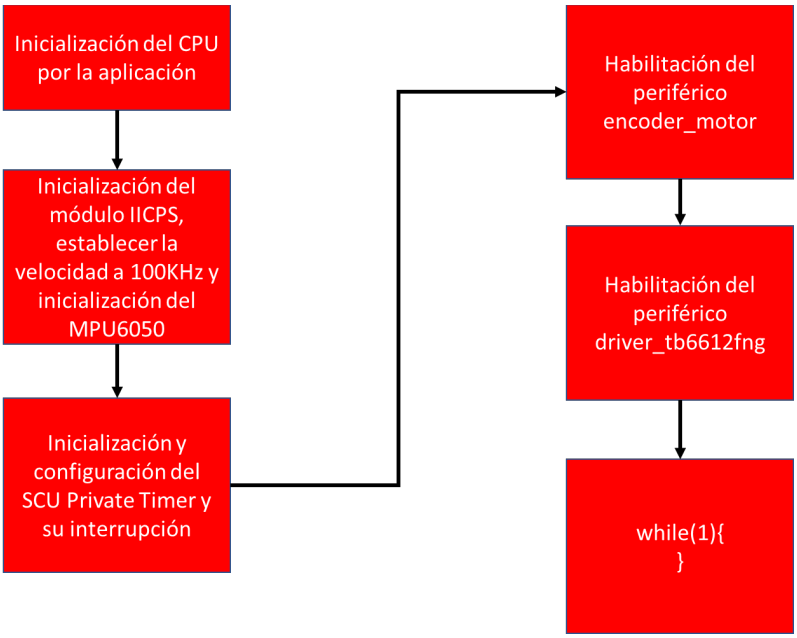


Figura 1-39 Proceso de inicialización

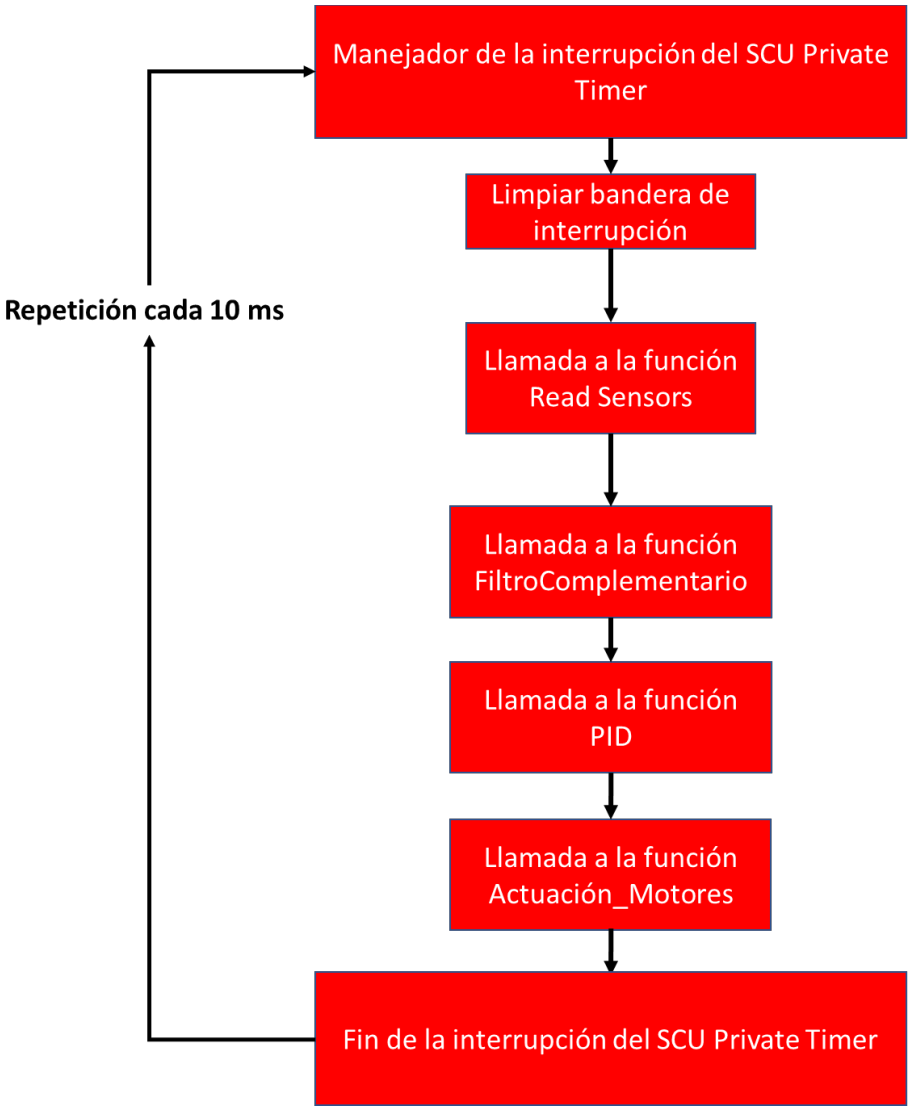


Figura 1-40 Esquema de funcionamiento del manejador de la interrupción

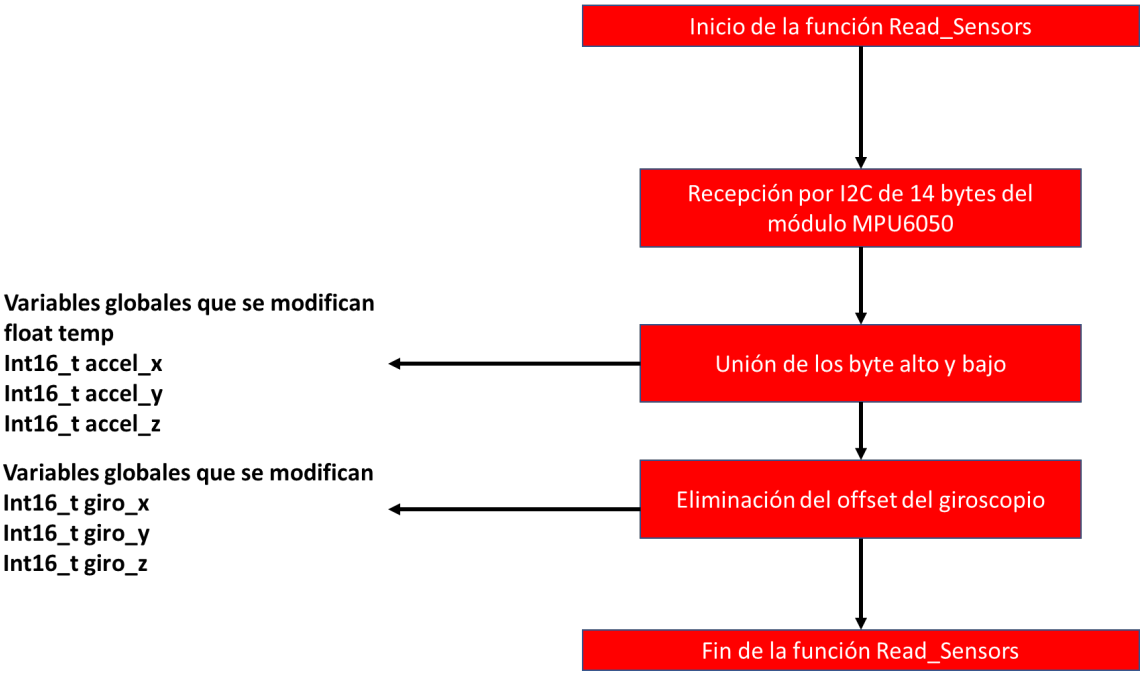


Figura 1-41 Función Read_Sensors

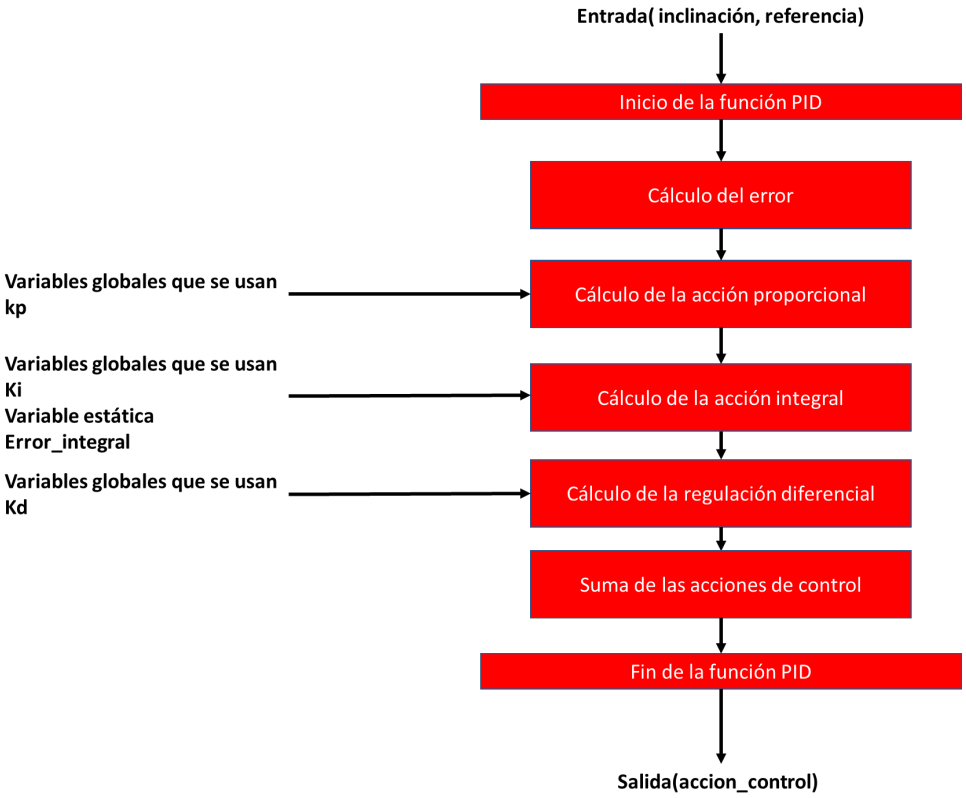


Figura 1-42 Función PID

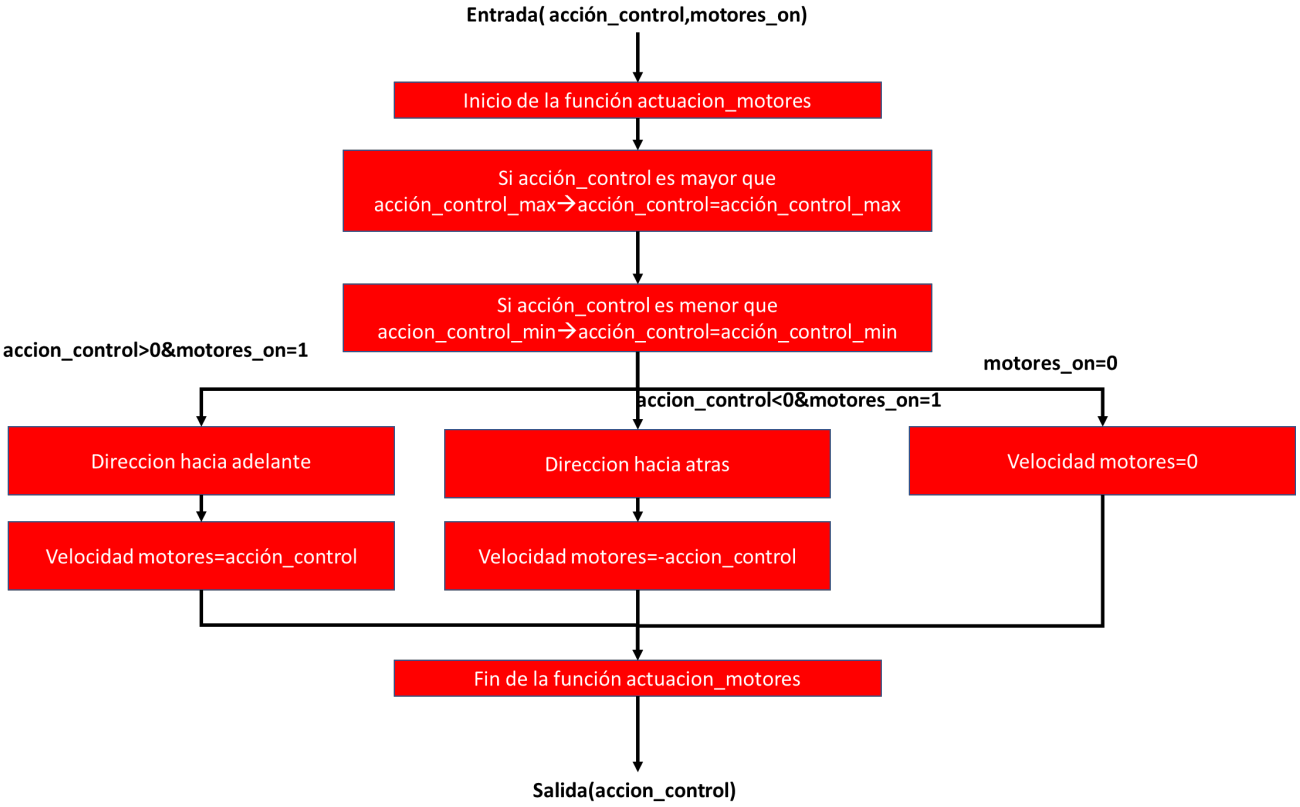


Figura 1-43 Esquema funcion accion_control

1.12 ORDEN DE PRIORIDAD

El orden de prioridad del proyecto es el siguiente:

1) Planos
2) Memoria
3) Pliego de condiciones
4) Presupuesto

UNIVERSIDAD DE LA RIOJA

TRABAJO DE FIN DE GRADO

TITULACIÓN: GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

CURSO: 2019/2020 **CONVOCATORIA:** SEPTIEMBRE

TÍTULO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR HARDWARE PARA UN SISTEMA INESTABLE

AUTOR: ROOSBEL VINICIO GUAYA VEGA

DIRECTOR/ES: JAVIER ESTEBAN VICUÑA MARTÍNEZ Y JAVIER RICO AZAGRA

DEPARTAMENTO: INGENIERÍA ELÉCTRICA

INDICE DE ANEXOS

2 ANEXOS

ANEXO I Modulo AXI-Lite driver TB6612FNG

ANEXO I - I Herramientas diseñadas para comunicarse desde el CPU con el módulo

Librería "driver_tb6612fng.h" para manejar el módulo:

```
#ifndef DRIVER_TB6612FNG_H
#define DRIVER_TB6612FNG_H


/****************** Include Files ********************/
#include "xil_types.h"
#include "xstatus.h"
#include "xil_io.h"
#define driver_tb6612fng_en 0
#define driver_tb6612fng_dirA 4
#define driver_tb6612fng_timA 8
#define driver_tb6612fng_dirB 12
#define driver_tb6612fng_timB 16

extern u32 driver_add; //Variable que contiene la dirección del periférico
/////
//Macro para escribir en los registros del periférico
//Se usa en todas las funciones para escribir en los registros
//////
#define DRIVER_TB6612FNG_mWriteReg(BaseAddress, RegOffset, Data) \
          Xil_Out32((BaseAddress) + (RegOffset), (u32)(Data))
/////
//Macro para leer los registros de los periféricos
/////
#define DRIVER_TB6612FNG_mReadReg(BaseAddress, RegOffset) \
    Xil_In32((BaseAddress) + (RegOffset))


////
//Modifica el valor de driver_add, que utilizan todas las funciones para la BaseAddress del periférico
////
void driver_motor_add(u32 num);
////
//Escribe en el registro en
//Tiene que habilitarse (en=0x1) el driver para que STBY=1 y el PWM funcione
////
void driver_motor_en(u32 num) ;
////////////////////////////////////////////////
//timA y timB modifican la PWM y pueden tomar valores desde 0 hasta 255
////////////////////////////////////////////////
////
//Escribe en el registro timA
////
void driver_motor_timA(u32 num);
////
//Escribe en el registro timB
////
void driver_motor_timB(u32 num);
////
//Escribe en los registros dirA y dirB
//Modifican los valores de AN1,AN2 y BN1,BN2
//Los cuales siempre son opuestos uno al otro (AN1 nunca=AN2)
////
void driver_motor_dir(u32 dirA,u32 dirB);
/////
//Función que se puede usar para realizar una prueba de si las comunicacion
//Entre el cpu y el periférico son correctas
//Básicamente escribe en los registros y lee si son iguales
///
XStatus DRIVER_TB6612FNG_Reg_SelfTest(void * baseaddr_p);


#endif // DRIVER_TB6612FNG_H
```

```
/*************************** Include Files ******************************/
#include "driver_tb6612fng.h"

/************************** Function Definitions ************************/

u32 driver_add;
void driver_motor_en(u32 num){
	DRIVER_TB6612FNG_mWriteReg(driver_add,driver_tb6612fng_en,num);
}
void driver_motor_add(u32 num){
	driver_add =num;
}
void driver_motor_timA(u32 num){
	//capado por ahora a 255
	num=0x000000FF&num;
	DRIVER_TB6612FNG_mWriteReg(driver_add,driver_tb6612fng_timA,num);
}
void driver_motor_timB(u32 num){
	//capado por ahora a 255
	num=0x000000FF&num;
	DRIVER_TB6612FNG_mWriteReg(driver_add,driver_tb6612fng_timB,num);
}
void driver_motor_dir(u32 dirA,u32 dirB){
	//capado por ahora a 255
	dirA=0x00000001&dirA;
	dirB=0x00000001&dirB;

	DRIVER_TB6612FNG_mWriteReg(driver_add,driver_tb6612fng_dirA,dirA);
	DRIVER_TB6612FNG_mWriteReg(driver_add,driver_tb6612fng_dirB,dirB);

}
```

ANEXO I - II Diseño de la lógica para y de las comunicaciones con AXI-Lite

Este periférico ha sido diseñado para comunicarse con el CPU mediante AXI-Lite y permitir al CPU manejar las señales que actúan sobre el control del módulo TB6612FNG.

Para ello tiene que ser capaz de generar señales PWM con capacidad de variar el tiempo en que la señal es positiva, para modificar el ciclo de trabajo. Esto se consigue con los **Bloque PWM base**, que reciben como entrada un valor de 0 a 255 y generan una PWM de ciclo de trabajo variable.

Como se van a controlar dos motores, en el bloque principal **Bloque principal de la lógica de usuario** se incluirán dos de estos bloques.

A continuación, en el siguiente apartado, se va a llevar a cabo el diseño de los bloques RTL en VHDL. Una vez finalizadas las simulaciones se procede a empaquetar el bloque lógico para unirlo a un bloque de comunicaciones AXI4-Lite que autogenera VIVADO.

El resultado final se representa en el siguiente esquema:

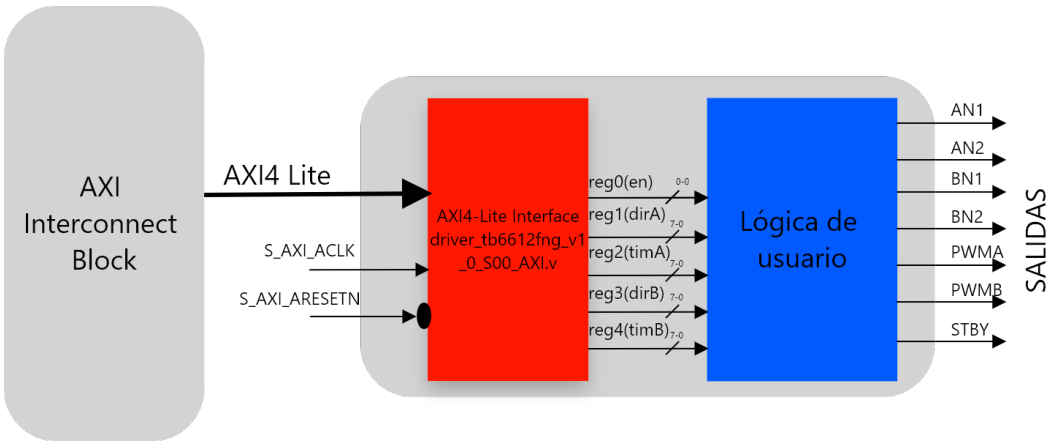


Figura 2-1 Esquema completo del módulo AXI4-Lite driver_tb6612fng

Como se puede ver en el esquema anterior, por la interfaz se reciben los valores de los registros y estos son pasados como señales al bloque Lógica de usuario, que es donde el módulo RTL se ha diseñado.

Estos registros son los siguientes. Se debe tener en cuenta que el offset se usará cuando se quiera dirigir con el CPU a este periférico. Si se escribe o lee a la dirección base el periférico más el offset, se puede acceder al registro que se quiera.

Offset	Nombre del registro	Funcionalidad
0x00	en	Habilita las PWM y habilita STBY
0x04	dirA	Controla AN1 y AN2 Si dirA=0: AN1=0 AN2=1 Si dirA=1: AN1=1 AN2=0
0x08	timA	Controla el tiempo en ON de la PWMA. Puede variar desde 0 hasta 255
0x0C	dirB	Controla BN1 y BN2 Si dirB=0: BN1=0 BN2=1 Si dirB=1: BN1=1 BN2=0
0x10	timB	Controla el tiempo en on de la PWMB. Puede variar desde 0 hasta 255

Tabla 2-1 Registros del periférico driver_tb6612fng

Para comunicarse con este periférico localizado en el PL, el CPU debe escribir a una dirección adecuada. Como se vio en apartado de Arquitectura de la ZYNQ-7000 SoC en el punto 1.9.2 del documento MEMORIA, si el CPU escribe a la dirección adecuada, su escritura aparecerá por los puertos GP0 o GP1.

La dirección adecuada se puede observar en la tabla de las direcciones del sistema del *ANEXO III*. En ella se observa que si se conecta al MGP0 las direcciones que permiten acceder a la lógica del PL son 0x4000_0000 – 0x7FFF_FFFF. Si se conecta al MGP1 el módulo, las direcciones válidas son 0x8000_0000 – 0xBFFF_FFFF.

Cuando queremos programar el CPU en SDK la dirección base del módulo está definida en xparameters.h con el nombre de **XPAR_DRIVER_TB6612FNG_0_S00_AXI_BASEADDR**.

El bloque debe ir conectado a una fuente de 10MHz, para generar pulsos que no sean demasiado rápidos para el driver.

Bloque PWM base

Para diseñar la lógica se creó primero un bloque que generara una señal PWM. Para ello se ha implementado un contador y un nivel (time_on). Mientras el contador sea menor que time_on la señal PWM=1, en caso contrario PWM=0.

El módulo en su representación de bloques es la siguiente:

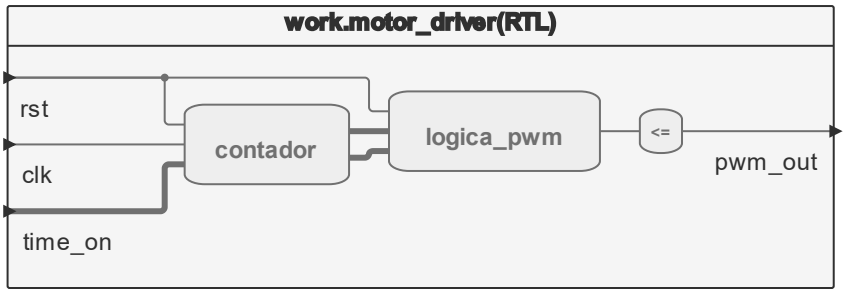


Figura 2-2 Diagrama de bloques de PWM base

Su código en VHDL es el siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity motor_driver is
	port(
		clk     : in  std_logic;
		rst     : in  std_logic;
		time_on : in  std_logic_vector(7 downto 0);
		pwm_out : out std_logic
	);
end entity motor_driver;

architecture RTL of motor_driver is
	signal pwm_aux : std_logic;
	signal counter : unsigned(7 downto 0);
	signal t_r:unsigned (7 downto 0);
begin

	contador:process(clk,time_on)
	begin
		if rst = '0' or (time_on /= std_logic_vector(t_r)) then
			counter <= (others => '0');
			t_r<=unsigned(time_on);
		else
			if (clk'event and clk = '1') then
				counter <= counter + 1;
			end if;

		end if;

	end process;

	logica_pwm:process(counter)
	begin
		if rst = '0' then
			pwm_aux <= '0';
		else
			if (t_r=255) then
				pwm_aux<='1';
			elsif (counter < t_r) then
				pwm_aux <= '1';
			else
				pwm_aux <= '0';
			end if;
		end if;
	end process;
	pwm_out <= pwm_aux;

end architecture RTL;
```

Bloque principal de la lógica de usuario

Este bloque contiene dos bloques generadores de PWM. Además se ha incluido un bloque lógico que se encarga de generar las señales AN1, AN2 y BN1, BN2 dependiendo de los valores de dirA y dirB respectivamente.

El valor de STBY, que habilitará o deshabilitará el funcionamiento de los motores, va unido a la señal en, la cual se encarga también de parar la generación de las señales PWM, como se puede observar del diagrama de bloques siguiente.

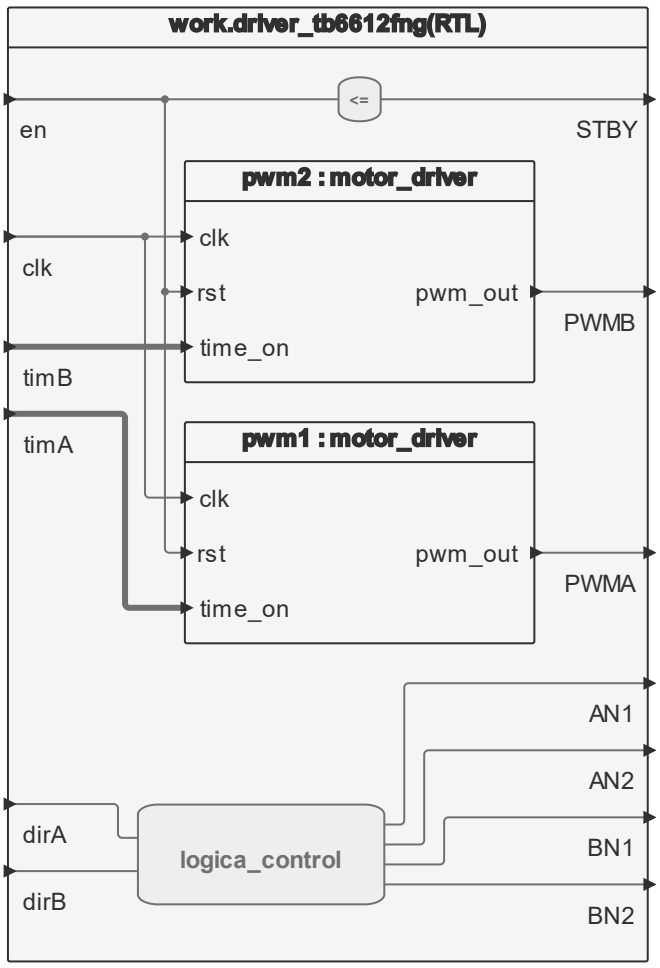


Figura 2-3 Diagrama de bloques del bloque lógico principal

Su código VHDL es el siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
--en activo a nivel alto
--en activa los contadores del pwm if counter<timA then PWM=1 else PWM=0
--stby<=en
entity driver_tb6612fng is
	port(
		clk  : in  std_logic;
		dirA : in  std_logic;
		dirB : in  std_logic;
		en   : in  std_logic;
		timA : in  std_logic_vector(7 downto 0);
		timB : in  std_logic_vector(7 downto 0);
		AN1  : out std_logic;
		AN2  : out std_logic;
		PWMA : out std_logic;
		BN1  : out std_logic;
		BN2  : out std_logic;
		PWMB : out std_logic;
		STBY : out std_logic
	);
end entity driver_tb6612fng;

architecture RTL of driver_tb6612fng is
	component motor_driver is
		port(
			clk     : in  std_logic;
			rst     : in  std_logic;
			time_on : in  std_logic_vector(7 downto 0);
			pwm_out : out std_logic
		);
	end component;
begin
	pwm1 : motor_driver
		port map(
			clk     => clk,             --           : in  std_logic;
			rst     => en,              --   : in  std_logic;
			time_on => timA,            --   : in  std_logic_vector(7 downto 0);
			pwm_out => PWMA             --   : out std_logic
		);
	pwm2 : motor_driver
		port map(
			clk     => clk,             --           : in  std_logic;
			rst     => en,              --   : in  std_logic;
			time_on => timB,            --   : in  std_logic_vector(7 downto 0);
			pwm_out => PWMB             --   : out std_logic
		);
	logica_control : process(dirA, dirB)
	begin
		if dirA = '1' then
			AN1 <= '0';
			AN2 <= '1';
		else
			AN1 <= '1';
			AN2 <= '0';
		end if;
		if dirB = '1' then
			BN1 <= '0';
			BN2 <= '1';
		else
			BN1 <= '1';
			BN2 <= '0';
		end if;
	end process;
	STBY <= en;
end architecture RTL;
```

Simulación de la lógica de usuario:

Se usa una testbench para comprobar que la lógica de dirA y dirB, hagan lo especificado. Por ejemplo: dirA=0, AN1=1 AN2=0.

Como se puede comprobar en la siguiente imagen, se ha cumplido.

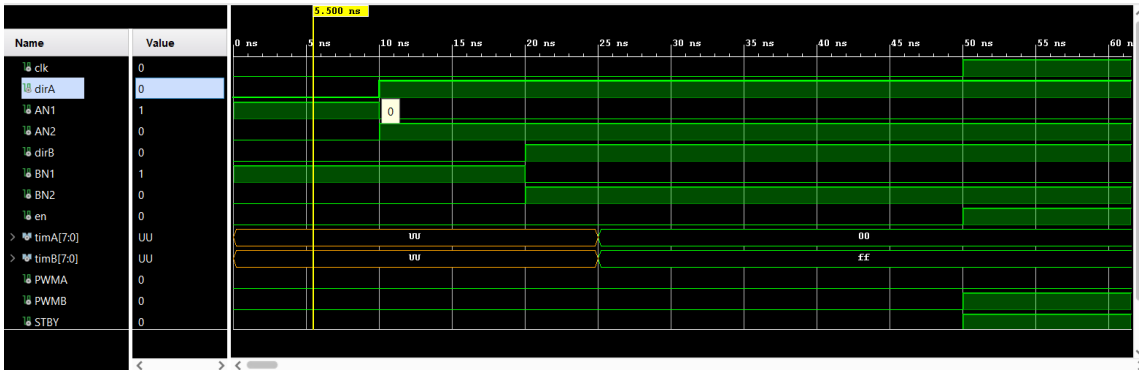


Figura 2-4 Simulación del bloque lógico principal 1

También se quiere comprobar el funcionamiento de time_on, se quiere ver si el tiempo en que la señal es positiva.

Como se puede comprobar en la siguiente imagen, la lógica cuenta bien los flancos ascendentes y pone adecuadamente el tiempo de encendido de la señal.

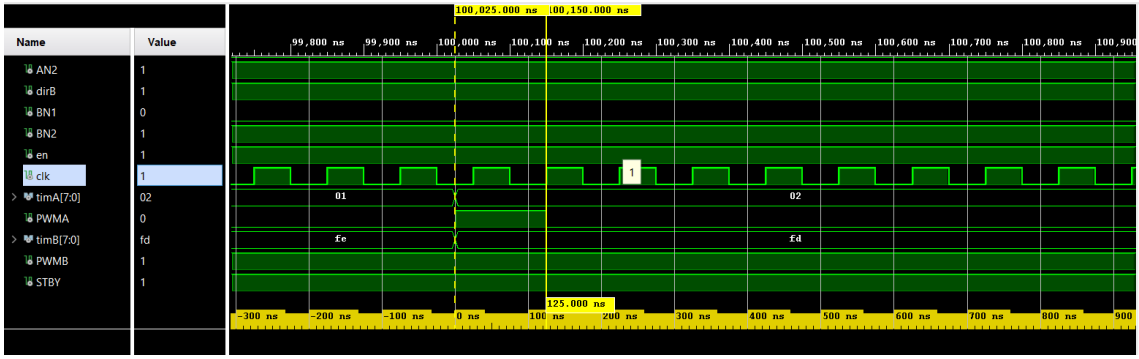


Figura 2-5 Simulación del bloque lógico principal 2

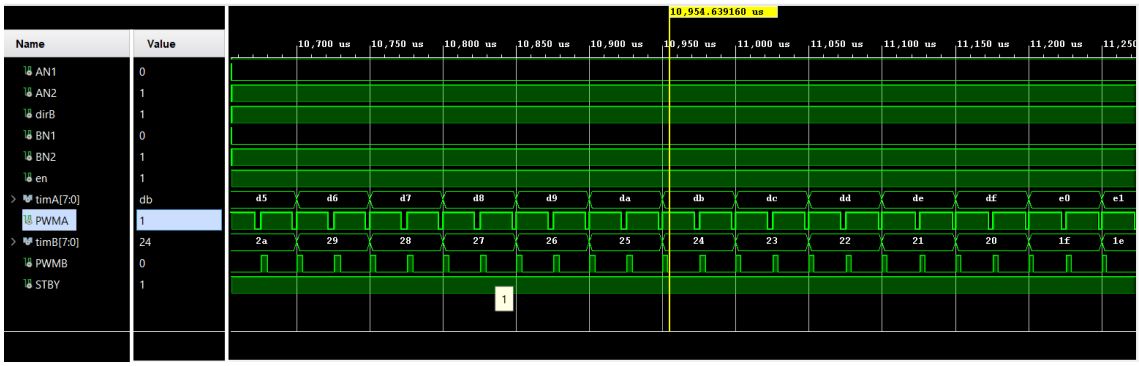


Figura 2-6 Simulación del bloque lógico principal 3

Comprobación de las comunicaciones:

Para la evaluación del módulo y para comprobar el correcto funcionamiento de las comunicaciones por AXI4 Lite y de la lógica programada se crea un proyecto y se lleva a cabo la instanciación de los bloques en IP INTEGRATOR.

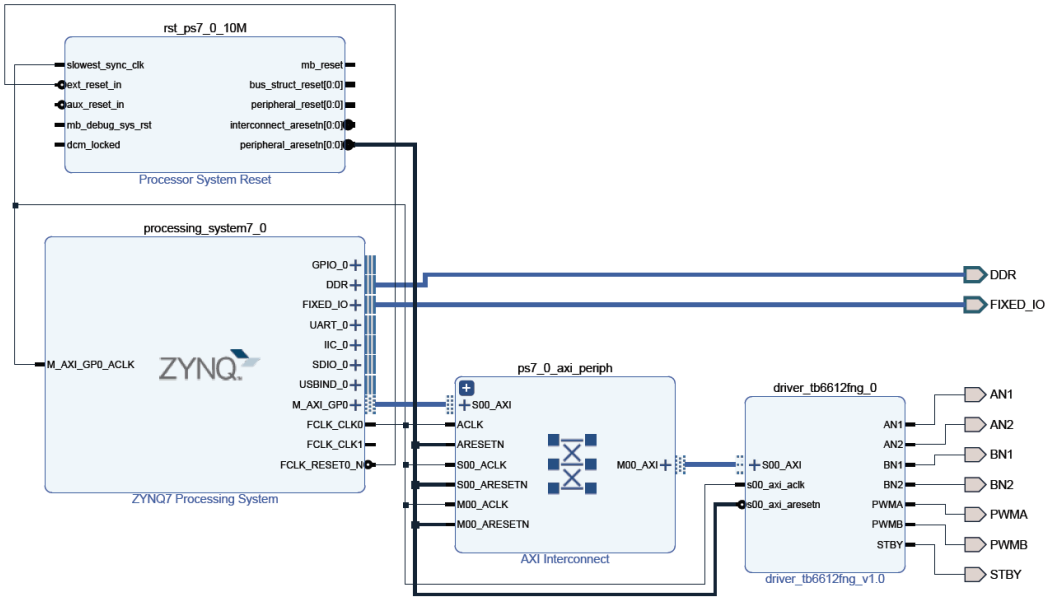


Figura 2-7 Diagrama de bloques para la comprobación del funcionamiento del driver_TB6612FNG

Se ha usado un analizador lógico para comprobar las señales. Se hará un bucle para el cual se irá incrementando el valor del tiempo en on (tima y tmB) hasta 255 y se tiene que ver cómo aumenta en proporción la señal.

Se usa el siguiente código en Xilinx SDK

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xil_io.h"
#include "driver_tb6612fng.h"
#include "sleep.h"
#include <stdbool.h>

u32 driver_add=XPAR_DRIVER_TB6612FNG_0_S00_AXI_BASEADDR;

int main()
{          u8 num=0;
           driver_motor_en(1);

           while(1){
                     for (int i=0;i<255;i+=1){
                               driver_motor_timA(i);
                               driver_motor_timB(i);
                               usleep(25);
                     }
                     num=num+1;
                     driver_motor_dir(0,num);

           }
    return 0;
}
```

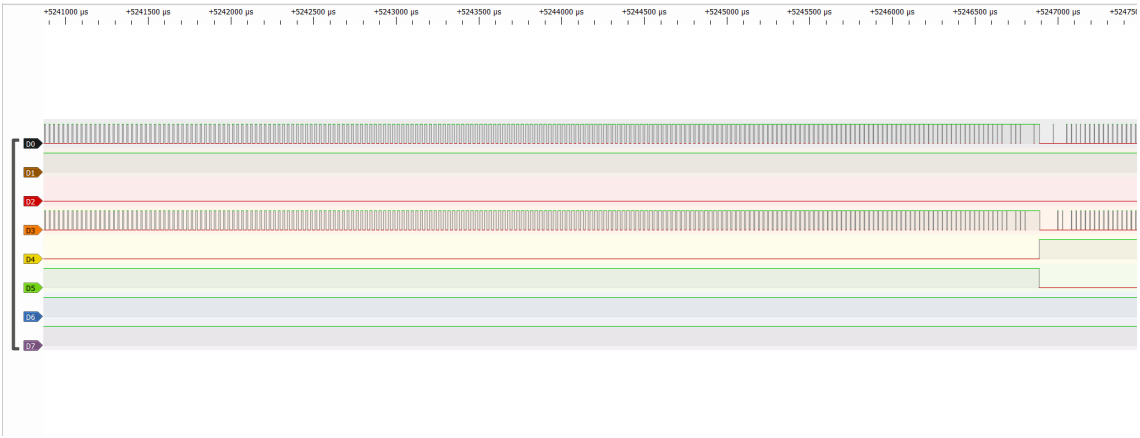


Figura 2-8 Comprobación del funcionamiento del PWMA y PWMB

Además, se debe comprobar las señales lógicas AN1 AN2 BN1 BN2 STBY.

Para ello se tiene en cuenta que la señal dirA se mantiene en 0, y la señal dirB cambia cada vez que se termina un bucle.

La señal STBY tiene que ser igual en todo momento a EN.

Este comportamiento se puede apreciar claramente en las siguientes imágenes.

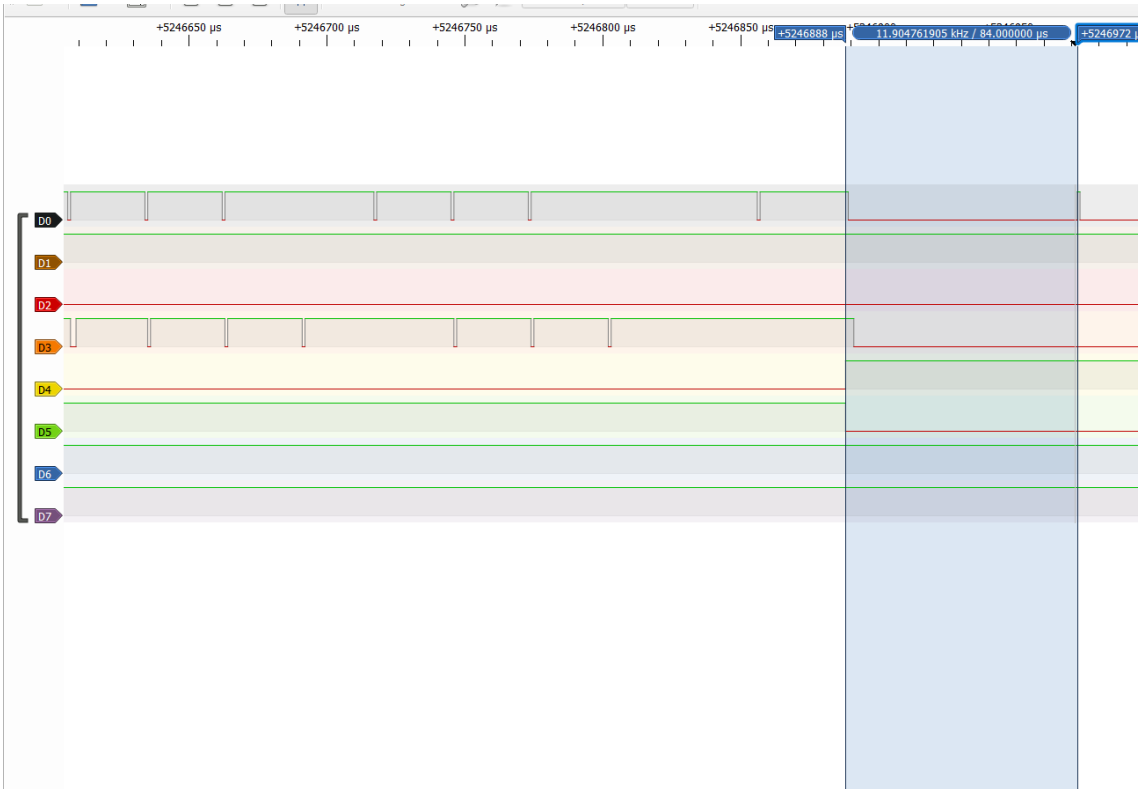


Figura 2-9 Comprobación de las señales de control 1

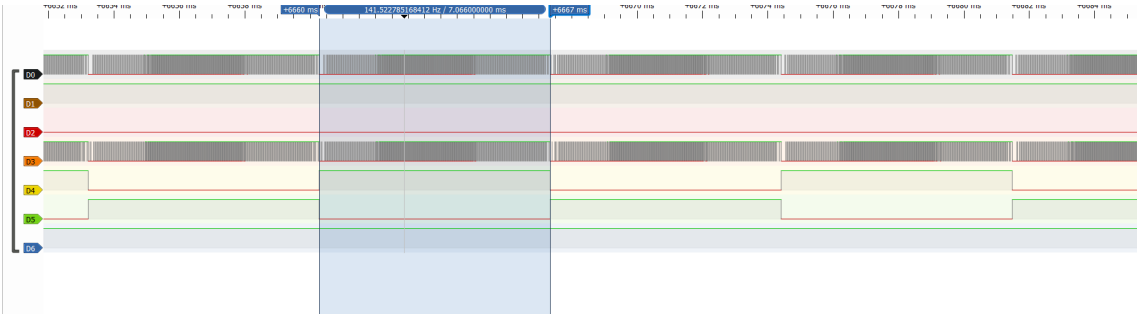


Figura 2-10 Comprobación de las señales de control 2

ANEXO II AXI-Lite encoders_motores

ANEXO II - I Herramientas diseñadas para comunicarse desde el CPU con el módulo Cabecera

```
#ifndef ENCODER_MOTOR_H
#define ENCODER_MOTOR_H


/****************** Include Files ********************/
#include "xil_types.h"
#include "xstatus.h"
#include "xil_io.h"
//Offset de los registros
//registros del encoder 1
#define ENCODER_MOTOR_S00_AXI_SLV_REG0_OFFSET 0// reg0(0 downto 0) enable1
#define ENCODER_MOTOR_S00_AXI_SLV_REG1_OFFSET 4//reg1(31 downto 0) pulsos
#define ENCODER_MOTOR_S00_AXI_SLV_REG2_OFFSET 8// reg2(31 downto 0)durPulsos
#define ENCODER_MOTOR_S00_AXI_SLV_REG3_OFFSET 12//reg3(1 downto 0) CCW&CW
//registros  del encoder 2
#define ENCODER_MOTOR_S00_AXI_SLV_REG4_OFFSET 16//reg4(0 downto 0) enable2
#define ENCODER_MOTOR_S00_AXI_SLV_REG5_OFFSET 20//reg5(31 downto 0) pulsos
#define ENCODER_MOTOR_S00_AXI_SLV_REG6_OFFSET 24//reg6(31 downto 0)durPulsos
#define ENCODER_MOTOR_S00_AXI_SLV_REG7_OFFSET 28//reg7(1 downto 0) CCW&CW


#define en_apagaPulso 0//apaga el contador de pulsos
#define en_enciendePulso 1//enciende el contador pulsos
#define en_hazreinicioPulso 2//con este valor el manejador de en hara que se ponga a 0 y luego a 1 en una sola instruccion
//Sera la direccion del modulo en el mapa de direcciones del sistema
//se debe declarar en donde se use el modulo
extern u32 dirEncoder;

//Macros para escribir y leer respectivamente en el los registros
//seran usados en las funiones
#define ENCODER_MOTOR_mWriteReg(BaseAddress, RegOffset, Data) \
                Xil_Out32((BaseAddress) + (RegOffset), (u32)(Data))


#define ENCODER_MOTOR_mReadReg(BaseAddress, RegOffset) \
  Xil_In32((BaseAddress) + (RegOffset))
```

```
/************************** Declaracion de funciones**************************/
/********* Encoder 1*******/
//funciones leer los registros del encoder 1


//depende del valor de value hace una reinicio del pulso pero lo hace volver a funcionar al siguiente pulso
//o lo apaga o lo enciende
//en_apagaPulso=0 en_enciendePulso=1 en_hazreinicioPulso=2
void write_encoder_motor_en_1(u32 value);
//lee el registro que cuenta los pulsos
void read_encoder_motor_pulsos_1(int32_t* value);
//lee el registro de duracion cada Pulso de los encoder
//siendo esta duracion la suma de 32 pulsos consecutivos
void read_encoder_motor_durPulsos_1(u32* value);
//lee el registro de direccion
void read_encoder_motor_dir_1(u32* dirCCW,u32* dirCW);
/********* Encoder 2*******/
//Funciones para leer los valores de los registros del encoder 2


void write_encoder_motor_en_2(u32 value);
void read_encoder_motor_pulsos_2(int32_t* value);
void read_encoder_motor_durPulsos_2(u32* value);
void read_encoder_motor_dir_2(u32* dirCCW,u32* dirCW);


////////////////////////////
//Funciones velocidad de las ruedas
////////////////////////////
//Estas funciones estan destinadas a ser usadas para obtener la velocidad de las ruedas cada 2,46 vueltas del encoder_motor
//O para obtener la velocidad cada 0.082 vueltas de la rueda
void read_encoder_motor_vel_rueda_1(float* rpm,float* rads);
void read_encoder_motor_vel_rueda_2(float* rpm,float* rads);
//La siguiente funcion puede ser usada en interrupciones de un temporizador
//Como entradas tiene el tiempo que ha tardado en ser activada la interrupcion
//Como salida tiene la velocidad media en el tiempo que ha tardado en activarse la interrupcion
void vel_rueda1(float time,float* rpm, float* rads);
void vel_rueda2(float time,float* rpm, float* rads);
#endif
```

Funciones

```
/***************************** Include Files *******************************/
#include "encoder_motor.h"
#define PI 3.14159265358979323846
#define durPulso50M 20E-9
/************************** Function Definitions ***************************/
/********* Encoder 1*******/
//funciones leer los registros del encoder 1
//depende del valor de value hace una reinicio del pulso pero lo hace volver a funcionar al siguiente pulso
//o lo apaga o lo enciende
//en_apagaPulso=0 en_enciendePulso=1 en_hazreinicioPulso=2
void write_encoder_motor_en_1(u32 value){
   if (value==en_hazreinicioPulso){
ENCODER_MOTOR_mWriteReg(dirEncoder,ENCODER_MOTOR_S00_AXI_SLV_REG0_OFFSET,0U);
ENCODER_MOTOR_mWriteReg(dirEncoder,ENCODER_MOTOR_S00_AXI_SLV_REG0_OFFSET,1U);

   }
   else if (value==en_enciendePulso ||value==en_apagaPulso )
   {
   ENCODER_MOTOR_mWriteReg(dirEncoder,ENCODER_MOTOR_S00_AXI_SLV_REG0_OFFSET,value);
   }
}
//lee el registro que cuenta los pulsos
void read_encoder_motor_pulsos_1(int32_t* value){
*value=ENCODER_MOTOR_mReadReg(dirEncoder,ENCODER_MOTOR_S00_AXI_SLV_REG1_OFFSET);
}
//lee el registro de duracion cada Pulso de los encoder
//siendo esta duracion la suma de 32 pulsos consecutivos
void read_encoder_motor_durPulsos_1(u32* value){

*value=ENCODER_MOTOR_mReadReg(dirEncoder,ENCODER_MOTOR_S00_AXI_SLV_REG2_OFFSET);
}
//lee el registro de direccion
void read_encoder_motor_dir_1(u32* dirCCW,u32* dirCW){
   u32 value;
value=ENCODER_MOTOR_mReadReg(dirEncoder,ENCODER_MOTOR_S00_AXI_SLV_REG3_OFFSET);
*dirCCW=(value>>1)&1; //&0010 el dirCCW esta en el bit 1
*dirCW=value&1; //&0001 el dirCW esta en el bit 0
}
```

```
/********* Encoder 2*******/
//Funciones para leer los valores de los registros del encoder 2


void write_encoder_motor_en_2(u32 value){
   if (value==en_hazreinicioPulso){

ENCODER_MOTOR_mWriteReg(dirEncoder,ENCODER_MOTOR_S00_AXI_SLV_REG4_OFFSET,0U);
ENCODER_MOTOR_mWriteReg(dirEncoder,ENCODER_MOTOR_S00_AXI_SLV_REG4_OFFSET,1U);

   }
   else if (value==en_enciendePulso ||value==en_apagaPulso )
   {
   ENCODER_MOTOR_mWriteReg(dirEncoder,ENCODER_MOTOR_S00_AXI_SLV_REG4_OFFSET,value);
   }


}
void read_encoder_motor_pulsos_2(int32_t* value){
*value=ENCODER_MOTOR_mReadReg(dirEncoder,ENCODER_MOTOR_S00_AXI_SLV_REG5_OFFSET);
}

void read_encoder_motor_durPulsos_2(u32* value){

*value=ENCODER_MOTOR_mReadReg(dirEncoder,ENCODER_MOTOR_S00_AXI_SLV_REG6_OFFSET);

}

void read_encoder_motor_dir_2(u32* dirCCW,u32* dirCW){
      u32 value;
value=ENCODER_MOTOR_mReadReg(dirEncoder,ENCODER_MOTOR_S00_AXI_SLV_REG7_OFFSET);
*dirCCW=(value>>1)&1; //&0010 el dirCCW esta en el bit 1
*dirCW=value&1; //&0001 el dirCW esta en el bit 0

}
```

```
////////////////////////////
//Funciones velocidad de las ruedas
////////////////////////////
//Estas funciones estan destinadas a ser usadas para obtener la velocidad de las ruedas cada 2,46 vueltas del encoder_motor
//O para obtener la velocidad cada 0.082 vueltas de la rueda
void read_encoder_motor_vel_rueda_1(float* rpm,float* rads){
float frec;
u32  durPulsos;
 durPulsos=ENCODER_MOTOR_mReadReg(dirEncoder,ENCODER_MOTOR_S00_AXI_SLV_REG2_OFFSET);
 frec= 1.0/(2.0*((durPulsos/32.0)*durPulso50M));//1/T  T=2*T_on T_on=durPulso*duracionunPulso50MHz
 *rpm=frec/(60.0*30.0*13.0);//1revrueda/min*60s/1min*30revmotor/1revrueda*13pulsos/1revmotor
 *rads=(*rpm)*2.0*PI/60.0;//rad/s
}
void read_encoder_motor_vel_rueda_2(float* rpm,float* rads){
float frec;
u32  durPulsos;
 durPulsos=ENCODER_MOTOR_mReadReg(dirEncoder,ENCODER_MOTOR_S00_AXI_SLV_REG6_OFFSET);
 frec= 1.0/(2.0*((durPulsos/32.0)*durPulso50M));//1/T  T=2*T_on T_on=durPulso*duracionunPulso50MHz
 *rpm=frec/(60.0*30.0*13.0);//1revrueda/min*60s/1min*30revmotor/1revrueda*13pulsos/1revmotor
 *rads=(*rpm)*2.0*PI/60.0;//rad/s
}
//La siguiente funcion puede ser usada en interrupciones de un temporizador
//Como entradas tiene el tiempo que ha tardado en ser activada la interrupcion
//Como salida tiene la velocidad media en el tiempo que ha tardado en activarse la interrupciï¿½n
void vel_rueda1(float time,float* rpm, float* rads){
                                int32_t pulsos;
                                float frec;
                                pulsos=ENCODER_MOTOR_mReadReg(dirEncoder,ENCODER_MOTOR_S00_AXI_SLV_REG1_OFFSET);
                                frec=pulsos/time;
                                 *rpm=frec*60.0/(30.0*13.0);//Entre 30 porq velocidad ruedas=velmotor/30
                                 //Entre 13 pq hay 13 pulsos por vuelta del motor
                                 //*60 para pasar de s a min
                                 *rads=(*rpm)*2.0*PI/60.0;//rad/s
                }
void vel_rueda2(float time,float* rpm, float* rads){
                                int32_t pulsos;
                                float frec;
                                pulsos=ENCODER_MOTOR_mReadReg(dirEncoder,ENCODER_MOTOR_S00_AXI_SLV_REG5_OFFSET);
                                frec=pulsos/time;
                                 *rpm=60*frec/(30.0*13.0);//Entre 30 porq velocidad ruedas=velmotor/30
                                 //Entre 13 pq hay 13 pulsos por vuelta del motor
                                 //*60 para pasar de s a min
                                 *rads=(*rpm)*2.0*PI/60.0;//rad/s
```

ANEXO II - II Diseño de la lógica de las comunicaciones con AXI-Lite

Este periférico ha sido diseñado para comunicarse con el CPU mediante AXI-Lite y permitir al CPU obtener los datos de los encoder, como es la velocidad, dirección y el número de vueltas que ha dado en una dirección.

Primero tiene que ser capaz de evitar contar señales de pulsos debidas al ruido que pueden hacer saltar la señal lógica. Estas señales son cortas en duración, demasiado cortas (menos de 10us) para ser producidas por el motor. La solución de este Antirrebote se desarrolla en el punto

Otro bloque se encarga de leer los pulsos y fijarse el sentido en que giran las ruedas dependiendo de cómo están desfasadas.

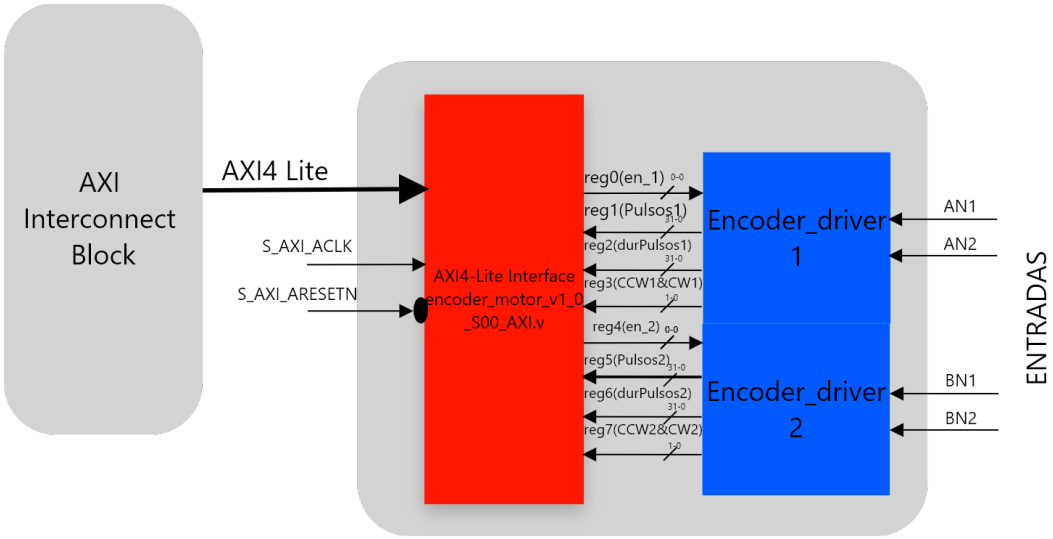
Estas señales las que salen del Antirrebote y las de dirección van al bloque que cuenta pulsos de manera ascendente si la rueda gira en sentido contrario a las agujas del reloj y si el sentido es el opuesto a cada pulso decrementa en una unidad.

Todos estos bloques se juntan para formar el bloque encoder_logic.

Para comunicarlo por AXI-Lite se van a instanciar dos de estos bloques uno encima de otro para leer los pulsos de los dos motores. Cada motor tiene dos encoder.

A continuación, en el siguiente apartado, se va a llevar a cabo el diseño de los bloques RTL en VHDL. Una vez finalizadas las simulaciones se procede a empaquetar el bloque lógico para unirlo a un bloque de comunicaciones AXI4-Lite que autogenera VIVADO. Es necesario modificar cómo trabaja con los registros pues el bloque los tiene como de lectura.

El resultado final se representa en el siguiente esquema:



Los registros son los siguientes:

Offset	Nombre del registro	Funcionalidad
0x00	En_1	Habilita el primer módulo encoder

0x04	Pulsos	Contiene la cuenta actual de cuantos pulsos se han dado en una dirección
0x08	durPulsos	Contiene la suma de 32 pulsos, se actualiza cada 32 pulsos
0x0C	CW_1&CW_1	Indica la dirección en que está girando la rueda
0x10	En_2	Habilita el segundo módulo encoder
0x14	Pulsos_2	Contiene la cuenta actual de cuantos pulsos se han dado en una dirección, del módulo encoder 2.
0x18	durPulsos_2	Contiene la suma de 32 pulsos, se actualiza cada 32 pulsos
0x1C	CW_2&CW_2	Indica la dirección en que está girando la rueda

Planteamiento del problema:

La siguiente figura

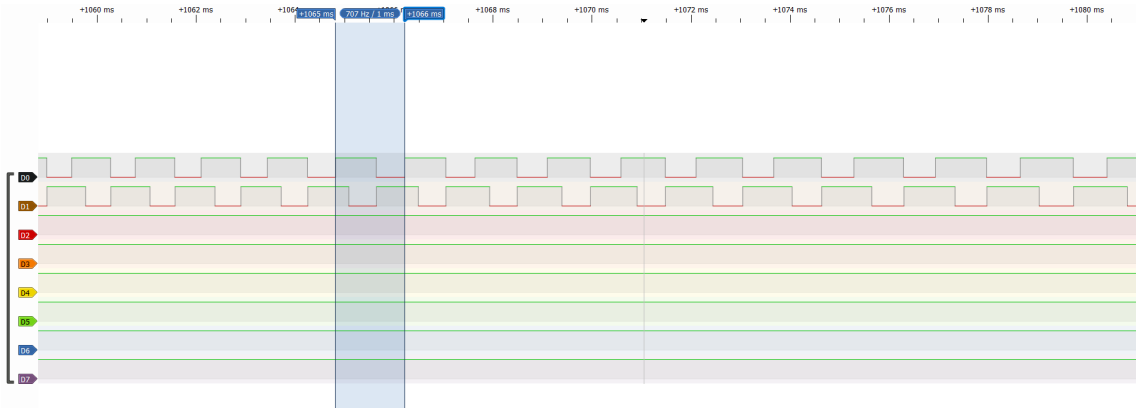


Figura 2-11 Funcionamiento de los encoder

Para conocer el comportamiento de los dos encoder de cada motor, se realizan para conocer cómo se comporta.

Estas pruebas ayudarán a especificar los requerimientos del módulo.

Lo primero que es necesario es leer la hoja de características que vino con el motor. En esta hoja se especifica lo siguiente:

Característica	**Valor**
Pulsos por vuelta por encoder	390 pulsos
Grados por pulso	0.923 grados/pulso
Radianes por pulso	0.01611 rad/pulso

Tabla 2-2 Diseño módulo para encoder, valores pulsos

Para las pruebas se ha puesto el motor a funcionar a plena potencia para conocer el periodo mínimo que tendrán los pulsos, así como de si se encuentran rebotes en la señal cuanto tiene que durar el antirrebotes para contrarrestar estos ruidos.

Los resultados son las siguientes figuras:

Con D0 conectado a B.

Con D1 conectado a A.

En la siguiente figura se ve que el tiempo mínimo que debe durar un pulso es alrededor de 690us.

También en esta como en la siguiente figura se puede apreciar cómo se puede utilizar los flancos para detectar el sentido de giro de las ruedas.

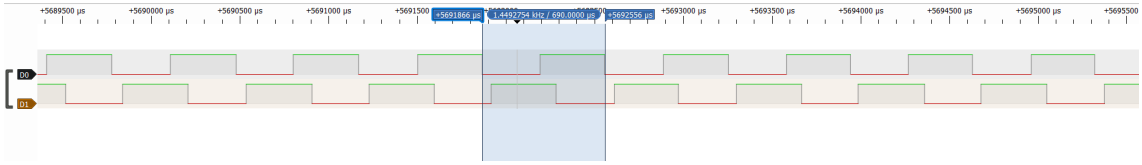


Figura 2-12 Rotación CCW

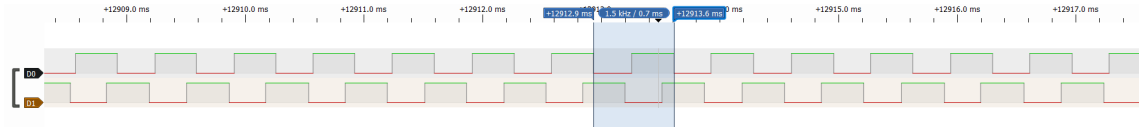


Figura 2-13 Rotación CW

Analizando las figuras anteriores, cuando el motor gira CCW (counter clock wise) la señal A adelanta a la del encoder B.

Y viceversa, cuando el motor gira CW la señal B adelanta a la A.

Esto se puede apreciar también en las siguientes tomas con el osciloscopio.

En las siguientes figuras el canal 1 corresponde con la señal que se obtiene del encoder B del motor.

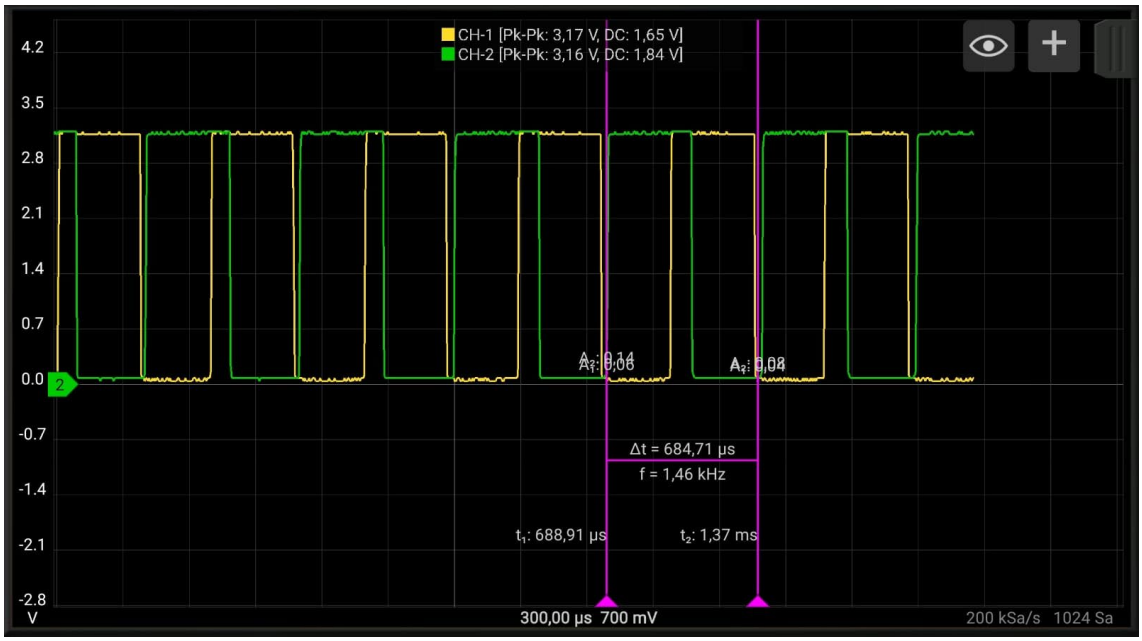


Figura 2-14 Señales obtenidas del osciloscopio de los dos encoder del motor, girando CCW

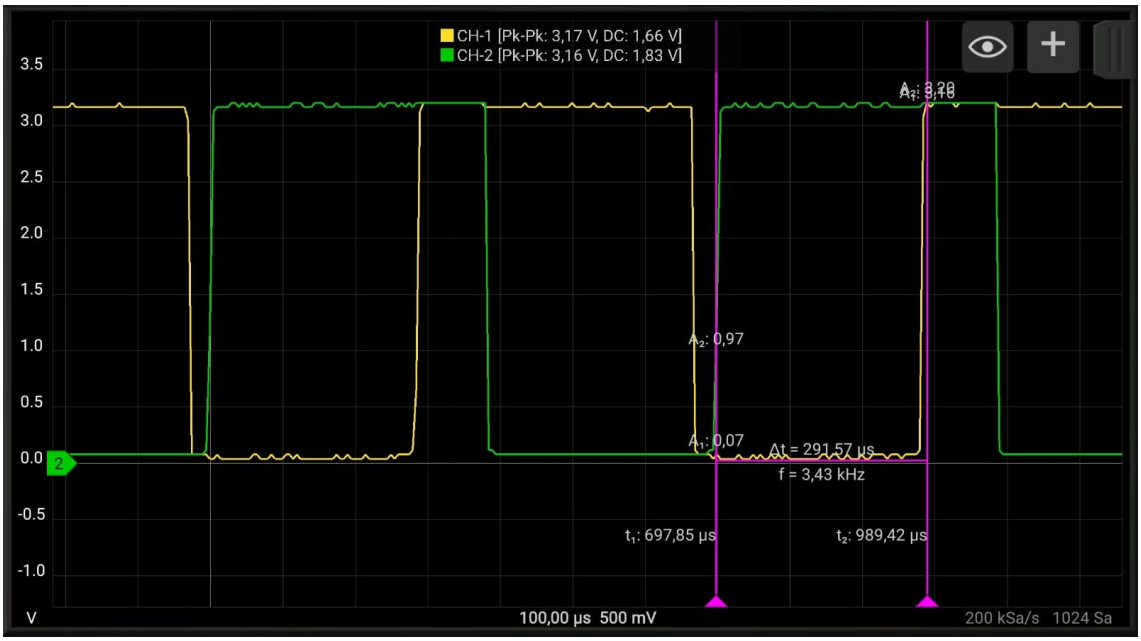


Figura 2-15 Señales obtenidas del osciloscopio de los dos encoder del motor, distancia mínima entre los pulsos

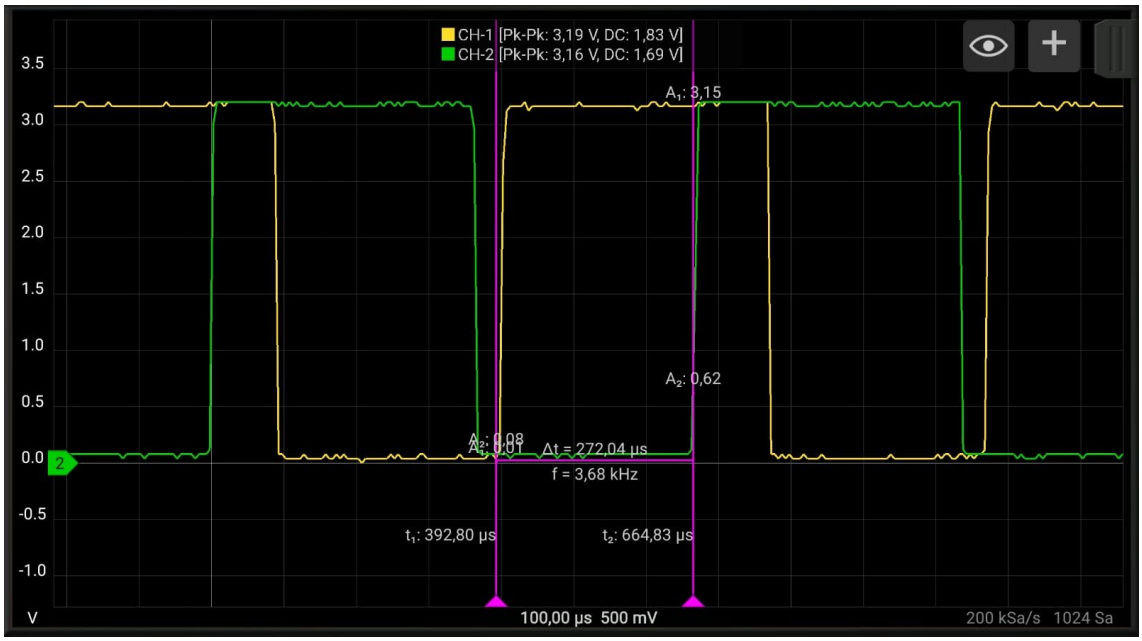


Figura 2-16 Señales obtenidas del osciloscopio de los dos encoder del motor, rotando CW

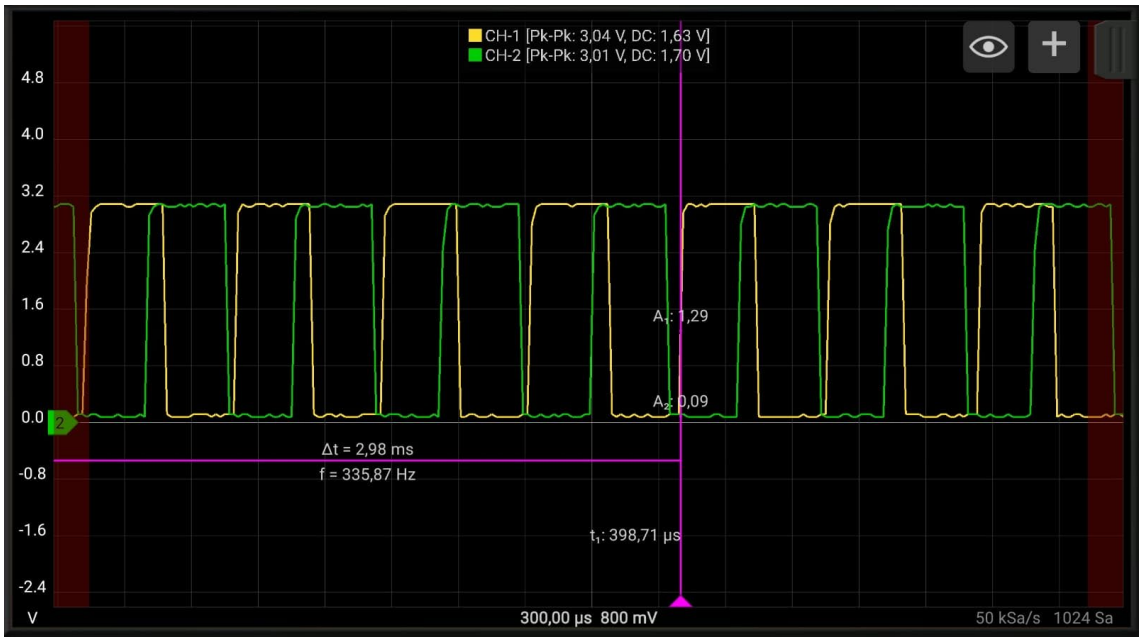


Figura 2-17 Señal obtenida de los encoder, después de atravesar el Level-Shifter

Cuando se alimentan los encoder con 3.3 V en vez de 5V empiezan a haber ciertos problemas en cuanto a la calidad de la señal, porque el analizador lógico empieza a detectar cosas que no son correctas. Se da cuenta de que no son posibes pues la duración de esos pulsos es demasiado pequeña:

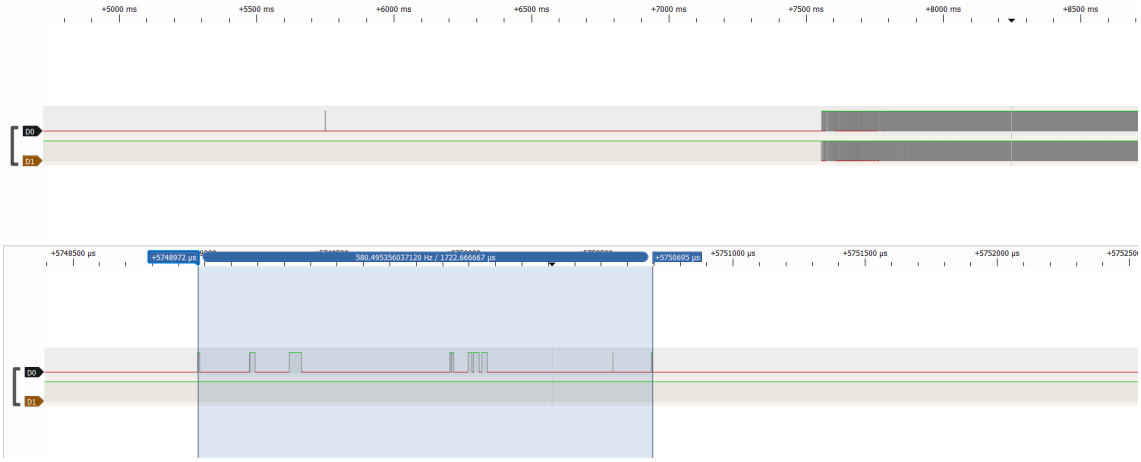


Figura 2-18 Ruido en la señal del encoder cuando está alimentado con 3.3V

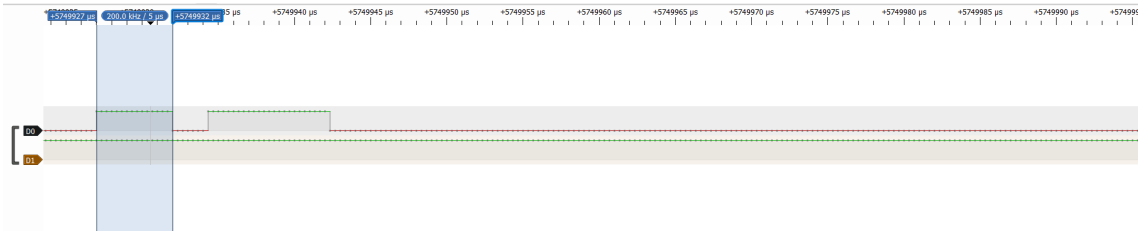


Figura 2-19 Ruido en la señal del encoder cuando está alimentado por 3.3V Zoom 1

Figura 2-20 Ruido en la señal del encoder cuando está alimentado por 3.3V Zoom 1

Como se puede apreciar en las figuras anteriores, si los encoder son a alimentados en vez de 5V, con 3.3V se van a detectar pulsos que no deberían detectarse.

Se debe tener en cuenta que cualquier variación en el nivel de la señal se detectarán a más velocidad incluso que el analizador lógico, pues la frecuencia del PL de la MiniZed va 300MHz.

Con todo esto llegamos a las siguientes observaciones y parámetros que se necesitarán para diseñar la unidad lógica que se comunicará con el CPU y recibirá las señales de los sensores:

Característica	
Duración mínima de un pulso de un encoder.	670 us
Duración mínima entre pulso y pulso (desfase)	272 us
Duración máxima de un rebote en la señal	10 us
Giro en CCW	A adelanta a B
Forma de detectar el sentido de giro	Cuando hay flanco ascendente en A mirar si la señal B es positiva (CW) o cero (CCW)
Pulsos por vuelta por encoder	390 pulsos

Grados por pulso	0.923 grados/pulso
Radianes por pulso	0.01611 rad/pulso
Detectar la velocidad	Contar tiempo que tarda en dar n pulsos

Tabla 2-3 Características del periférico driver-tb6612fng

Diseño del 100áximum100ote:

Este bloque tiene como objetivo eliminar los pulsos con menos de 10us de duración.

Para conseguir esto, y no haya un conteo erróneo de pulsos, es necesario diseñar un 100áximum100ote que se encargue de eliminar estos pequeños pulsos, que para los motores no será posible alcanzar.

Para ello se ha diseñado la máquina de estados que se muestra a continuación:

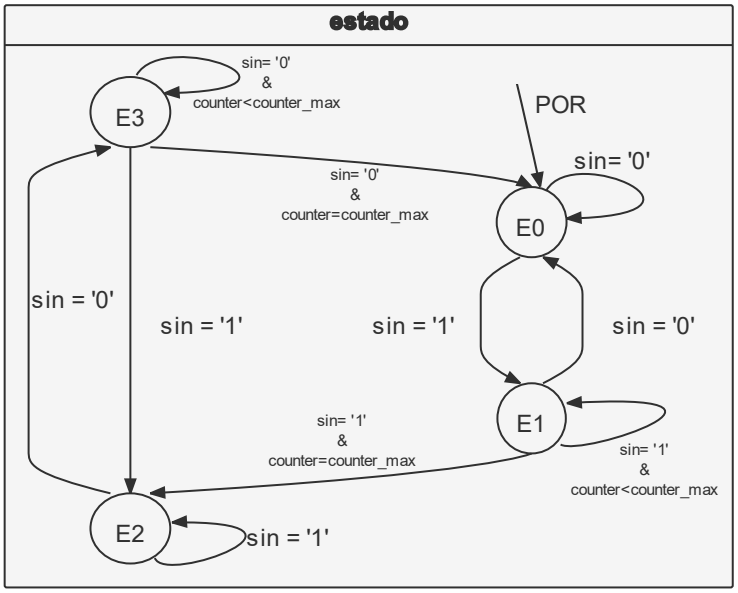


Figura 2-21 Maquina de estados del 100áximum100ote

En el código a continuación se puede observar cómo el genérico D_MAX_DURACION, tendrá significado en cuanto a cuantos pulsos del clk a 50 MHz debe durar el pulso como mínimo.

Si ponemos un valor de 500 los pulsos de menos de 10us no afectarán a la señal.

El código VHDL este bloque es el siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity antirrebote is
	generic(
		--pulsos a 50MHz de duracion maxima del rebote
		D_MAX_REBOTE : integer := 500
	);
	port(
		clk_50M : in  std_logic;
		rst     : in  std_logic;
		sin     : in  std_logic;
		sout    : out std_logic
	);
end entity antirrebote;
architecture RTL of antirrebote is
	signal counter            : integer range 0 to D_MAX_REBOTE;
	type es is (E0, E1, E2, E3);
	signal estado, estado_sig : es;
	signal counter_en         : std_logic;
begin
	CONTADOR : process(clk_50M)
	begin
	if clk_50M='1' and clk_50M'event then
		if rst = '0' or counter_en = '0' then
			counter <= 0;
		else
			counter <= counter + 1;
		end if;
	end if;
	end process;
	FSM_CAMESTADO : process(clk_50M)
	begin
	if clk_50M='1' and clk_50M'event then
		if rst = '0' then
			estado <= E0;
		else
			estado <= estado_sig;
		end if;
	end if;
	end process;
	FSM_ESTADO_SIG : process(sin, counter, estado)
	begin
		estado_sig <= estado;
		case (estado) is
			when E0 =>
				sout       <= '0';
				counter_en <= '0';
				if sin = '1' then
					estado_sig <= E1;
				else
					estado_sig <= E0;
				end if;

			when E1 =>
				sout       <= '0';
				counter_en <= '1';
				if sin = '0' then
					estado_sig <= E0;
				else
					if (counter < D_MAX_REBOTE) then
						estado_sig <= E1;
					else
						estado_sig <= E2;
					end if;
				end if;

			when E2 =>
				sout       <= '1';
				counter_en <= '0';
				if sin = '1' then
					estado_sig <= E2;
				else
					estado_sig <= E3;
				end if;

			when E3 =>
				sout       <= '1';
				counter_en <= '1';
				if sin = '1' then
					estado_sig <= E2;
				else
					if counter < D_MAX_REBOTE then
						estado_sig <= E3;
					else
						estado_sig <= E0;
					end if;
				end if;
		end case;
	end process;

end architecture RTL;
```

Para saber si están funcionando bien se genera una testbench donde se intenta ver cuándo la señal se pondrá a nivel alto, y si detecta los pulsos menores de 10us en este caso cuando se programa el genérico D_MAX_REBOTE a 500. (Que equivale a 500 pulsos de CLK a 50 MHz).

El código es el siguiente y lo que básicamente hace es generar pulsos cada vez mayores en sin para ver cuando el 102áximum102ote deja pasar señales.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity testbench_antirrebote is
end entity testbench_antirrebote;

architecture RTL of testbench_antirrebote is
	component antirrebote
		generic(
			--pulsos a 50MHz de duracion maxima del rebote
			D_MAX_REBOTE : integer := 500
		);
		port(
			clk_50M : in  std_logic;
			rst     : in  std_logic;
			sin     : in  std_logic;
			sout    : out std_logic
		);
	end component;
	signal clk_50M : std_logic;
	signal rst     : std_logic;
	signal sin     : std_logic;
	signal sout    : std_logic;

begin
	antirre : antirrebote
		generic map(
			--pulsos a 50MHz de duracion maxima del rebote
			D_MAX_REBOTE => 500
		)
		port map(
			clk_50M => clk_50M,		--: in  std_logic;
			rst     => rst,			--  : in  std_logic;
			sin     => sin,			--  : in std_logic;
			sout    => sout			--  : out std_logic
		);

	clock : process
	begin
		--50MHz clk
		clk_50M <= '0';
		wait for 10 ns;
		clk_50M <= '1';
		wait for 10 ns;
	end process;
	stimulus : process
	begin
		rst <= '0', '1' after 50 ns;
		for I in 1 to 1000 loop
			sin <= '0';
			for D in 0 to I loop
				wait for 10 ns;
			end loop;
			sin <= '1';
			for D in 0 to I loop
				wait for 10 ns;
			end loop;
		end loop;
		wait for 1 us;
		for I in 1000 to 10000 loop
			sin <= '0';
			for D in 0 to I loop
				wait for 10 ns;
			end loop;
			sin <= '1';
			for D in 0 to I loop
				wait for 10 ns;
			end loop;
		end loop;
		wait;
	end process;
end architecture RTL;
```

En las siguientes figuras se puede observar cómo el Antirrebote no deja pasar pulsos menores de 10us, y aunque retrasa la señal 10us, la duración del pulso se mantiene igual pues hay un retraso de 10us tanto en la subida de flanco como en la caída.

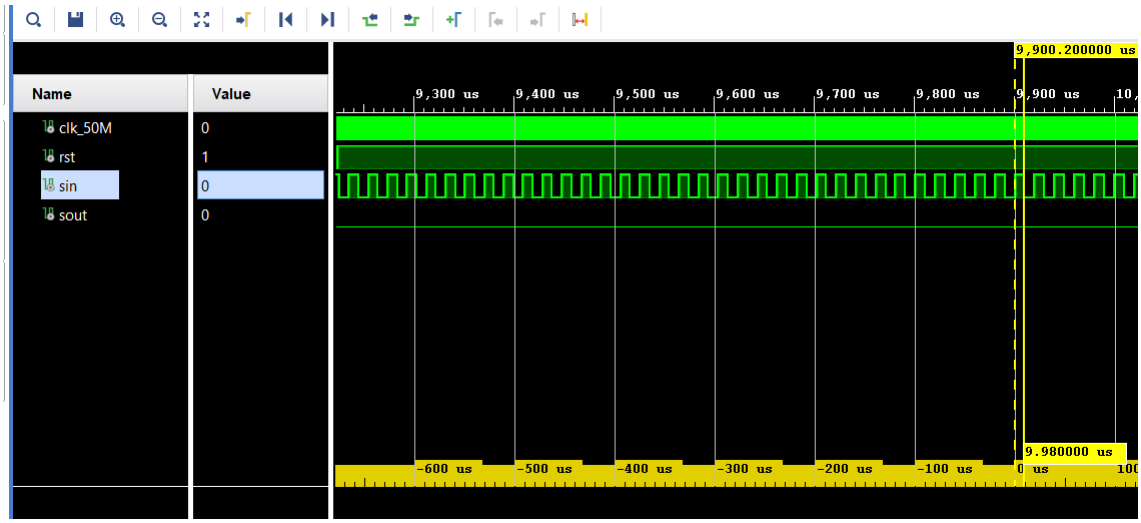


Figura 2-22 Simulación Antirrebote 1

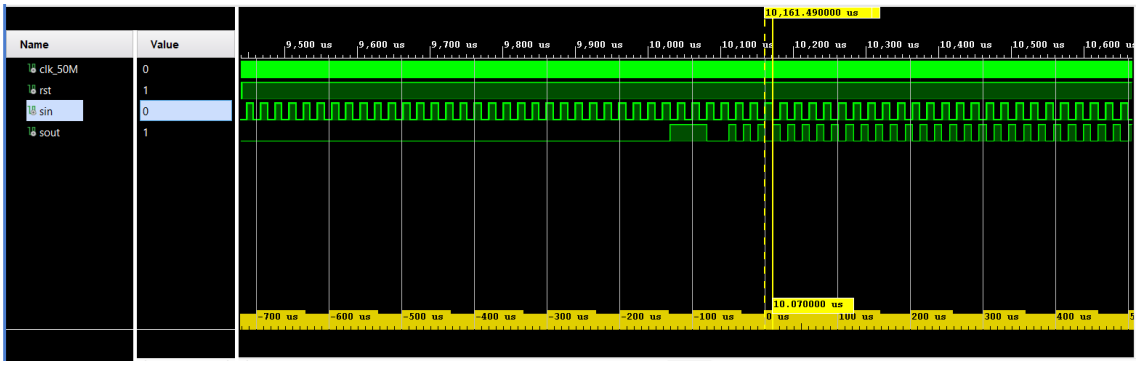


Figura 2-23 Simulación Antirrebote 2

Bloque de detección de la dirección de la rueda

Para detectar la dirección de las ruedas se observan las figuras del osciloscopio donde se ve que, si se detecta el flanco ascendente de la señal del encoder A, si en ese momento la señal B está a nivel bajo, entonces el motor gira en CCW y si está a nivel alto el motor gira en CW.

Por ello se diseña un bloque que detecte pulsos y mantenga la señal encendida, de que ha ocurrido un pulso, durante un CLK.

Para ello se usa el siguiente código:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity encoder_direction is
          port(
                    clk_50M : in  std_logic;
                    rst     : in  std_logic;
                    sinA    : in  std_logic;
                    sinB    : in  std_logic;
                    dirCCW  : out std_logic;
                    dirCW   : out std_logic
          );
end entity encoder_direction;

architecture RTL of encoder_direction is
          signal dirCCW_aux, dirCW_aux : std_logic;
          signal sinA0_input : std_logic;
          signal sinA1_input : std_logic;
          signal risingEdgeSinA:std_logic;
begin
                    --detector de flancos
detector_flancos : process(clk_50M,rst)
begin
  if rst='0' then
    sinA0_input<= '0';
    sinA1_input<= '0';
  elsif(rising_edge(clk_50M)) then
    sinA0_input           <= sinA;
    sinA1_input           <= sinA0_input;
end if;
end process detector_flancos;
risingEdgeSinA <= (not sinA1_input) and sinA0_input;

          deteccion_direccion : process(rst, risingEdgeSinA)
          begin
                    if risingEdgeSinA='1' then
                    if rst = '0' then
                              dirCCW_aux <= '0';
                              dirCW_aux  <= '0';
              else
                                        if sinB = '0' then
                                                  dirCCW_aux <= '1';
                                                  dirCW_aux  <= '0';

                                        else
                                                  dirCCW_aux <= '0';
                                                  dirCW_aux  <= '1';
                                   end if;
        end if;
       end if;
          end process;
          dirCCW <= dirCCW_aux;
          dirCW  <= dirCW_aux;
end architecture RTL;
```

Se genera una testbench para conocer el funcionamiento de la lógica donde el objetivo es saber si detectará o no la dirección cuando una señal está desfasada de la otra.

Este es el código:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity testbench_encoder_direction is
end entity testbench_encoder_direction;

architecture RTL of testbench_encoder_direction is

	component encoder_direction
		port(
			clk_50M     : in  std_logic;
			rst         : in  std_logic;
			sinA        : in  std_logic;
			sinB        : in  std_logic;
			dirCCW : out std_logic;
			dirCW    : out std_logic
		);
	end component;
	signal clk_50M     : std_logic;
	signal rst         : std_logic;
	signal sinA        : std_logic;
	signal sinB        : std_logic;
	signal dirCCW : std_logic;
	signal dirCW    : std_logic;

begin
	dir_detector : encoder_direction
		port map(
			clk_50M     => clk_50M,     --: in  std_logic;
			rst         => rst,         --: in  std_logic;
			sinA        => sinA,        --: in  std_logic;
			sinB        => sinB,        --: in  std_logic;
			dirCCW => dirCCW, --: out std_logic;
			dirCW    => dirCW     --: out std_logic
		);
	clock : process
	begin
		--50MHz clk
		clk_50M <= '0';
		wait for 10 ns;
		clk_50M <= '1';
		wait for 10 ns;
	end process;
	stimulus : process
	begin
		rst  <= '0', '1' after 50 ns;
		sinA <= '0';
		sinB <= '0';
		wait for 50 ns;
		primero_sinA : for I in 0 to 255 loop
			sinA <= '1';
			wait for 10 ns;
			sinB <= '1';
			wait for 1 us;
			sinA <= '0';
			wait for 10 ns;
			sinB <= '0';
			wait for 1 us;
		end loop;
		primero_sinB : for I in 0 to 255 loop
			sinB <= '1';
			wait for 10 ns;
			sinA <= '1';
			wait for 1 us;
			sinB <= '0';
			wait for 10 ns;
			sinA <= '0';
			wait for 1 us;
		end loop;

		wait;
	end process;
end architecture RTL;
```

La simulación se lleva a cabo en Vivado:

Como se puede ver en la figura cuando el flanco de A es detectado y la señal de B se detecta como a nivel bajo, entonces el sentido es CCW (en sentido contrario a las agujas del reloj).

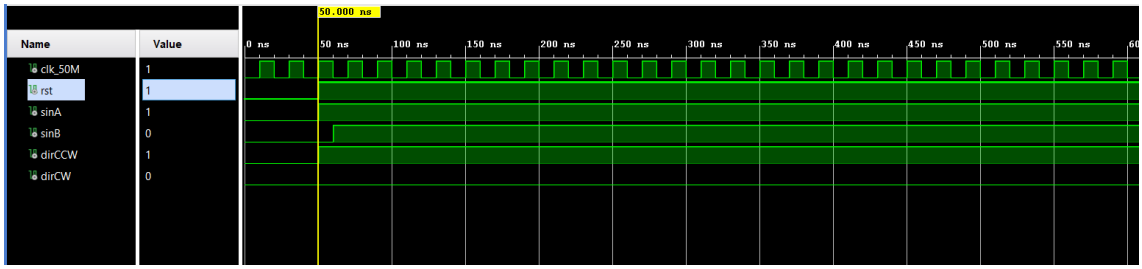


Figura 2-24 Simulación modulo sentido de giro 1

En la siguiente figura, el orden de las señales cambia, y ahora la señal B adelanta a la A y entonces la señal de CW pasa a nivel alto.

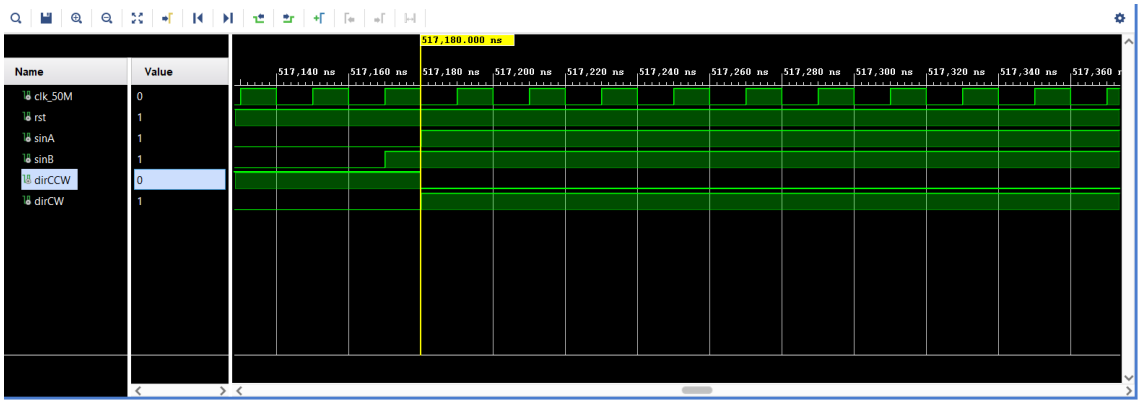


Figura 2-25 Simulación sentido de giro 2

Bloque contador de frecuencia:

Este bloque será usado para medir la velocidad del motor. De lo que se encarga este bloque es de medir la duración de un pulso, hacer la media durante n ciclos (es obligatorio que n sea potencia de dos) y sacar el valor en la salida.

El código VDHL es el siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity contador_frecuencia is
          generic(
                    --valor máximo 20
                    NUM_PULSOS_MEDIAR : integer := 32 -- cuenta el número de pulsos de los que se van
                    --a hacer media para contar la frecuencia
          );
          port(
                    clk_50M    : in  std_logic;
                    rst        : in  std_logic;     --resetea el sistema, activo a nivel bajo
                    sinA       : in  std_logic;
                    dirCCW     : in  std_logic;
                    dirCW      : in  std_logic;
                    durPulsos  : out std_logic_vector(31 downto 0); --duracion pulsos en 20ns
                    pulsos     : out std_logic_vector(31 downto 0) -- suma el numero de pulsos (dirCCW=1-> ascendente)
          );
end entity contador_frecuencia;

architecture RTL of contador_frecuencia is
          signal pulsos_aux   : signed(31 downto 0);
          signal cont_frec    : unsigned(31 downto 0);
          signal cont_frec_en : std_logic;
          signal regAux       : unsigned(31 downto 0);
          signal selec_reg: unsigned(31 downto 0);

          signal durPulso_aux : std_logic_vector(31 downto 0);
          signal sinA0_input : std_logic;
          signal sinA1_input : std_logic;
          signal risingEdgeSinA:std_logic;
          signal fallingEdgeSinA:std_logic;
begin

pulsos <= std_logic_vector(pulsos_aux);

contador_pulsos_sinA:process (clk_50M)
begin
   if clk_50M='1' and clk_50M'event then
      if rst='0' then
         pulsos_aux <= (others => '0');
      elsif risingEdgeSinA='1' then
         if dirCCW='1' then
            pulsos_aux <= pulsos_aux + 1;
         elsif dirCW='1' then
            pulsos_aux <= pulsos_aux - 1;
         end if;
      end if;
   end if;
end process;

contador_frecuencia_50M : process(clk_50M)
          begin
          if clk_50M='1' and clk_50M'event then
      if cont_frec_en='0' then
         cont_frec <= (others => '0');
     else
          cont_frec<=cont_frec+1;
      end if;
   end if;
          end process;

          --detector de flancos
detector_flancos : process(clk_50M,rst)
begin
  if rst='0' then
    sinA0_input<= '0';
    sinA1_input<= '0';
  elsif(rising_edge(clk_50M)) then
    sinA0_input           <= sinA;
    sinA1_input           <= sinA0_input;
end if;
end process detector_flancos;
risingEdgeSinA <= (not sinA1_input) and sinA0_input;
fallingEdgeSinA <= sinA1_input and (not sinA0_input);
```

```
        suma_duracion_NUMPULSOS_pulsos : process( rst,risingEdgeSinA,fallingEdgeSinA)
        begin
                if rst = '0' then
                        cont_frec_en <= '0';
                        regAux       <= (others => '0');
                        durPulso_aux<=(others=>'0');
                        selec_reg<=(others=>'0');
                else

                        if risingEdgeSinA='1' then
                                cont_frec_en <= '1';
                        end if;
                        if fallingEdgeSinA='1' then

                                if selec_reg < (NUM_PULSOS_MEDIAR) then
                                        regAux    <= regAux + cont_frec;
                                        selec_reg<=selec_reg+1;
                                else
                                        durPulso_aux <=std_logic_vector(regAux);
                                        selec_reg<=(others=>'0');
                                    regAux<=(others=>'0');
                                end if;
                        cont_frec_en <= '0';
                        end if;
                end if;
        end process;

        durPulsos <= durPulso_aux;
end architecture RTL;
```

Para comprobar el funcionamiento del bloque se diseña una testbench que se encarga de comprobar que el tiempo medido del pulso en on es correcto. Para ello genera señales de 5 us de duración. La primera vez se darán 3 pulsos con las señales CW=’0’ y CCW=’0’, un valor no válido y por tanto el módulo no debe contar pulsos. En la siguiente fase de la testbench se generan 500 pulsos de duración de 5us y con dirección de giro CCW (CW=’0’ CCW=’1’) y después 600 pulsos de duración 5us cada uno, pero con la dirección inversa (CW=’1’ CCW=’0’).

Por tanto, los valores que se deben ver en la simulación son 250 en el valor de duración, primero no cuenta nada, después el valor del número de pulsos va aumentando hasta 500 y luego al cambiar la dirección, va disminuyendo hasta -100.

La señal pulsos llevará la cuenta del número de pulsos válidos que ha detectado desde que se ha iniciado el módulo.

La señal durPulsos contiene la suma de 32 pulsos. Esta señal cambia cada 32 flancos descendentes de la señal sinA.

La señal regAux acumula la duración de los pulsos y su reinicio está sincronizado con la actualización del valor en durPulsos.

Como se puede ver en los primeros 3 pulsos no cuenta el contador.

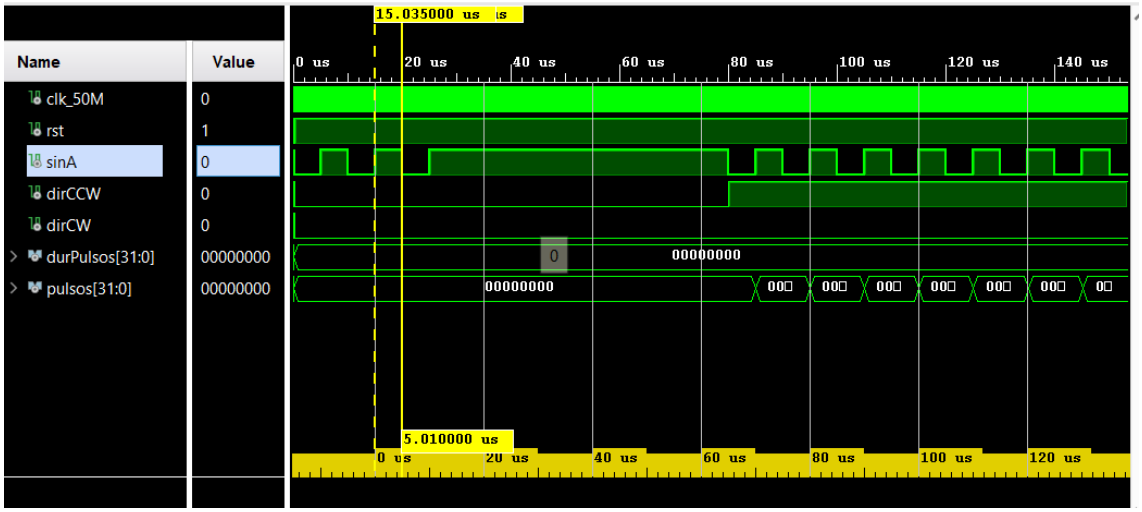


Figura 2-26 Simulación del bloque contador frec 1

Como se puede ver en la siguiente figura, el contador (pulsos) aumenta, pero durPulsos que contiene la duración de los 32 últimos pulsos no se actualiza hasta que hayan pasado los 32 pulsos.

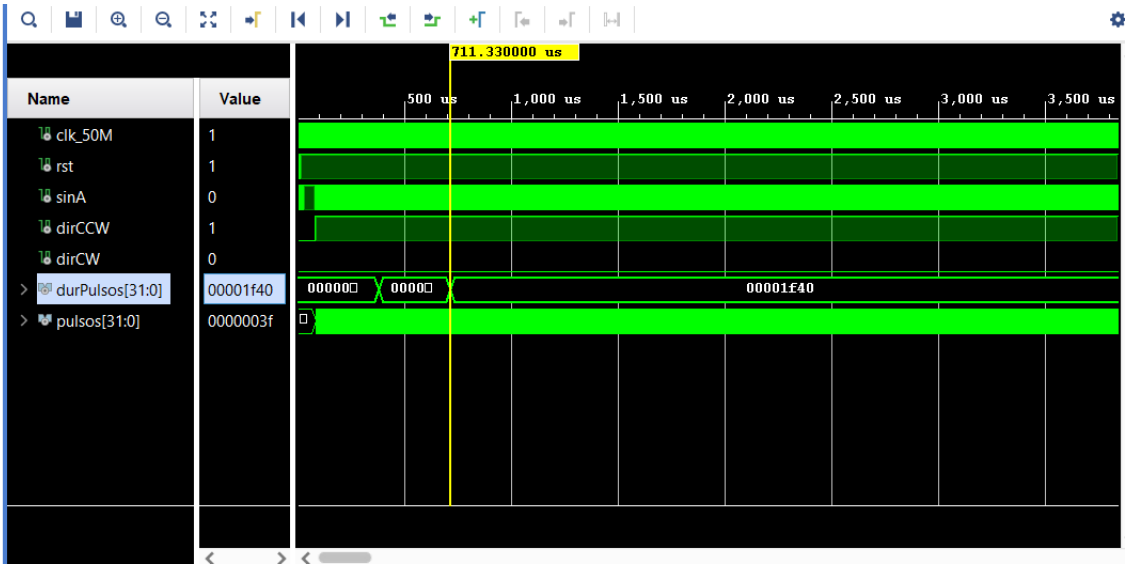


Figura 2-27 Simulación del bloque contador frec 2

Haciendo zoom, se puede ver cómo el contador de 50MHz cuenta 250·32=8000 pulsos porque la duración de los pulsos generados es de 5 us.

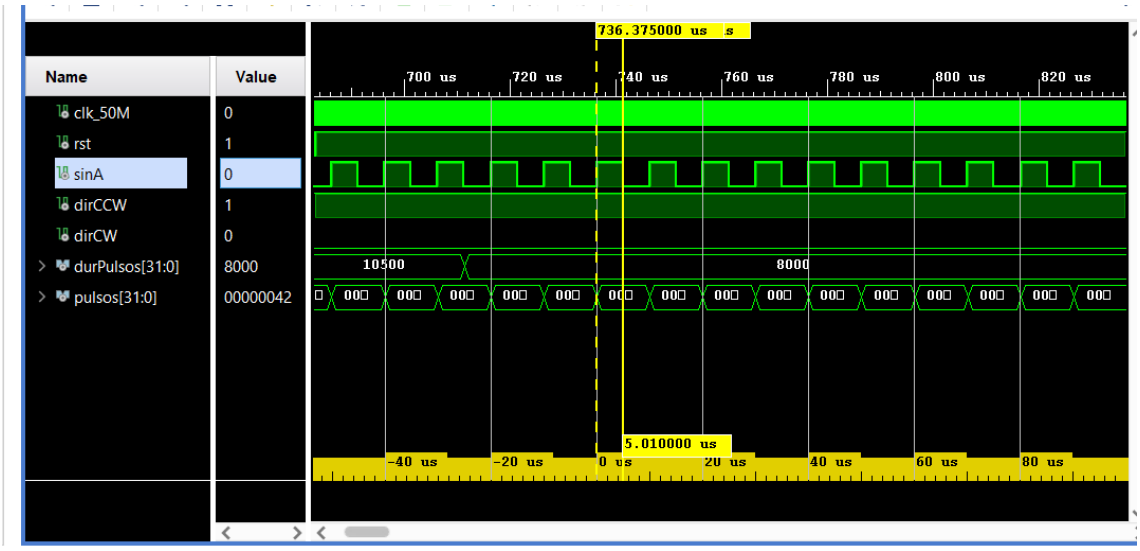


Figura 2-28 Simulación del bloque contador frec 3

Por último, se mira que al final de la duración de los 500 pulsos en sinA, el contador ha contado 500 pulsos, y empieza a decrecer cuando se cambia la dirección (dirCCW, dirCW)

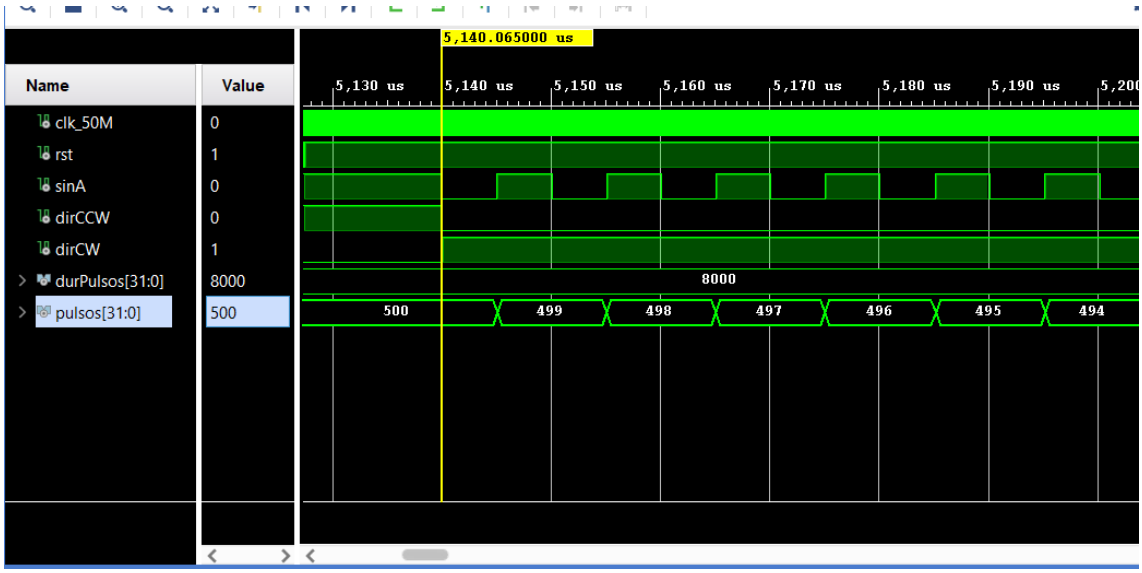


Figura 2-29 Simulación del bloque contador frec 4

Como se puede ver el contador de pulsos contará hasta -100, comprobando el funcionamiento en el sentido opuesto del módulo.

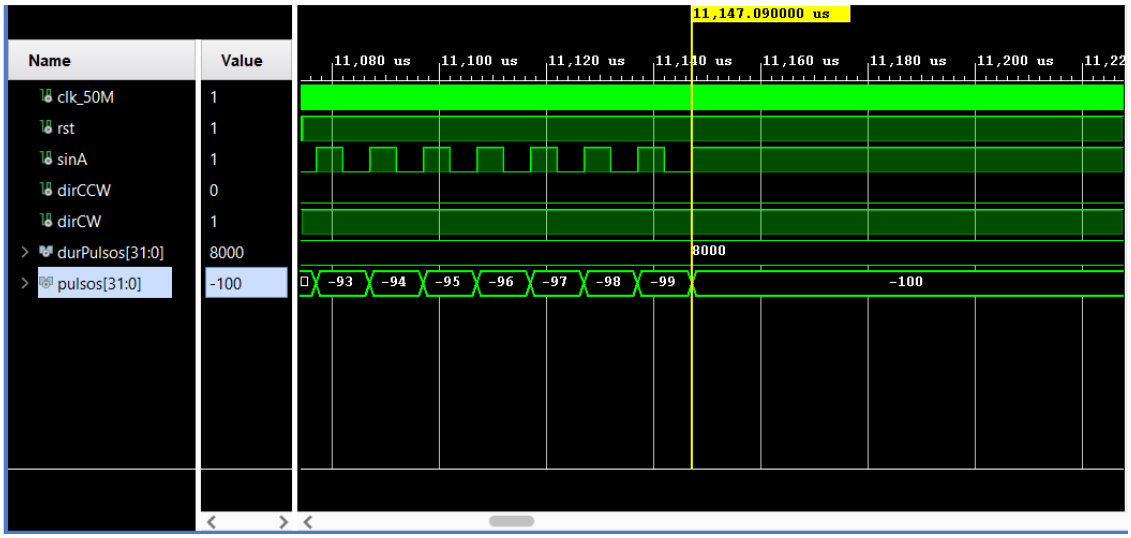


Figura 2-30 Simulación contrador frec 5

El código de la testbench es el siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity testbench_contador_frec is
end entity testbench_contador_frec;

architecture RTL of testbench_contador_frec is
	component contador_frecuencia
		generic(
			--valor máximo 20
			NUM_PULSOS_MEDIAR : integer := 20 -- cuenta el número de pulsos de los que se van
			--a hacer media para contar la frecuencia
		);
		port(
			clk_50M    : in  std_logic;
			rst        : in  std_logic; --resetea el sistema, activo a nivel bajo
			sinA       : in  std_logic;
			dirCCW     : in  std_logic;
			dirCW      : in  std_logic;
			durPulsos  : out std_logic_vector(31 downto 0); --duracion pulsos en 20ns
			pulsos     : out std_logic_vector(31 downto 0) -- suma el numero de pulsos (dirCCW=1-> ascendente)
		);
	end component;
	signal clk_50M    : std_logic;
	signal rst        : std_logic;      --resetea el sistem
	signal sinA       : std_logic;
	signal dirCCW     : std_logic;
	signal dirCW      : std_logic;
	signal durPulsos  : std_logic_vector(31 downto 0);
	signal pulsos     : std_logic_vector(31 downto 0);
begin
	modulo_logico : contador_frecuencia
		generic map(
			--valor máximo 20
			NUM_PULSOS_MEDIAR => 32     -- cuenta el número de pulsos de los que se van
			--a hacer media para contar la frecuencia
		)
		port map(
			clk_50M    => clk_50M,      --: in  std_logic;
			rst        => rst,          --: in  std_logic; --resetea el sistema, activo a nivel bajo
			sinA       => sinA,         --: in  std_logic;
			dirCCW     => dirCCW,       --: in  std_logic;
			dirCW      => dirCW,        --: in  std_logic;
			durPulsos  => durPulsos,    --: out std_logic_vector(31 downto 0); --duracion pulsos en 20ns
			pulsos     => pulsos        --: out std_logic_vector(31 downto 0) -- suma el numero de pulsos (dirCCW=1-> ascendente)
		);

	clock : process
	begin
		--50MHz clk
		clk_50M <= '0';
		wait for 10 ns;
		clk_50M <= '1';
		wait for 10 ns;
	end process;
	stimulus : process
	begin
		wait for 5 ns; --desincronizar el clk de la inicializacion de las señales
		--puesta en marcha de los reset (Desactivarlos)
		rst <= '0', '1' after 50 ns;
		--declaración de la dirección

		--con las dos valiendo a 0 no debe contar
		dirCCW<='0';
		dirCW<='0';

		--señales de 5us de duracion
		--tantos pulsos como diga el fin del loop
		--debe haber 3 pulsos
		for I in 1 to 3 loop
			sinA <= '0';
			for D in 0 to 500 loop
				wait for 10 ns;
			end loop;
			sinA <= '1';
			for D in 0 to 500 loop
				wait for 10 ns;
			end loop;
		end loop;
		wait for 50 us;
		--declaración de la dirección

		--Al ponerlo en dirección CCW el contador debe aumentar
		dirCCW<='1';
		dirCW<='0';

		--señales de 5us de duracion
		--tantos pulsos como diga el fin del loop

		--en este caso debe haber 500 pulsos
		--el valor de durPulsos debe ser 250
		for I in 1 to 500 loop
			sinA <= '0';
			for D in 0 to 500 loop
				wait for 10 ns;
			end loop;
			sinA <= '1';
			for D in 0 to 500 loop
				wait for 10 ns;
			end loop;
		end loop;
		wait for 50 us;
```

```
                        --declaración de la dirección

                        --Al ponerlo en dirección CW el contador debe disminuir
                        dirCCW<='0';
                        dirCW<='1';

                        --señales de 5us de duracion
                        --tantos pulsos como diga el fin del loop

                        --en este caso debe haber 500 pulsos
                        --el valor de durPulsos debe ser 250
                        for I in 1 to 600 loop
                                    sinA <= '0';
                                    for D in 0 to 500 loop
                                                wait for 10 ns;
                                    end loop;
                                    sinA <= '1';
                                    for D in 0 to 500 loop
                                                wait for 10 ns;
                                    end loop;
                        end loop;
                        wait for 50 us;

                        wait;
            end process;
end architecture RTL;
```

Bloque de encoder_logic:

Este será el módulo englobará a los anteriores. En este módulo se incluye el antirrebote para llevar a cabo la eliminación de señales de menos de 10 us. También se incluye un módulo de detección del sentido de giro y un módulo de detección de la duración de los pulsos de la señal de entrada. El módulo que detecta la duración del pulso, también hace una suma de los valores. Es un bloque genérico y por tanto se puede aumentar las pulsaciones al CLK de 50MHz que hace media.

El módulo se activa cuando el rst=’1’.

La siguiente imagen es el diagrama de bloques del módulo:

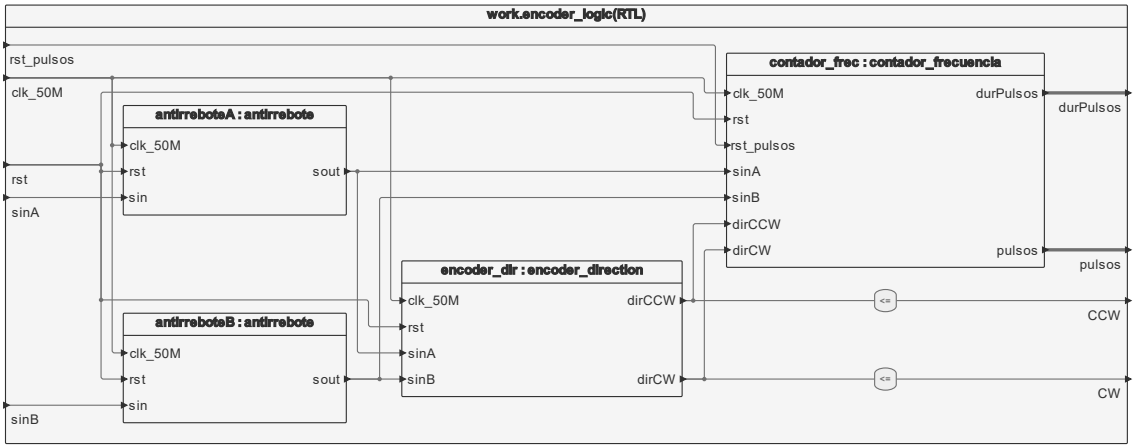


Figura 2-31 Diagrama de bloques módulo encoder_logic

El código del bloque es el siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity encoder_logic is
	port(
		clk_50M    : in  std_logic;
		rst        : in  std_logic;
		sinA       : in  std_logic;
		sinB       : in  std_logic;
		pulsos     : out std_logic_vector(31 downto 0);
		durPulsos  : out std_logic_vector(31 downto 0);
		CCW        : out std_logic;
		CW         : out std_logic
	);
end entity encoder_logic;

architecture RTL of encoder_logic is
	component antirrebote
		generic(
			--pulsos a 50MHz de duracion maxima del rebote
			D_MAX_REBOTE : integer := 500
		);
		port(
			clk_50M : in  std_logic;
			rst     : in  std_logic;
			sin     : in  std_logic;
			sout    : out std_logic
		);
	end component;

	component contador_frecuencia
		generic(
			--valor máximo 20
			NUM_PULSOS_MEDIAR : integer := 32 -- cuenta el número de pulsos de los que se van
			--a hacer media para contar la frecuencia
		);
		port(
			clk_50M    : in  std_logic;
			rst        : in  std_logic; --resetea el sistema, activo a nivel bajo
			sinA       : in  std_logic;
			dirCCW     : in  std_logic;
			dirCW      : in  std_logic;
			durPulsos  : out std_logic_vector(31 downto 0); --duracion pulsos en 20ns
			pulsos     : out std_logic_vector(31 downto 0) -- suma el numero de pulsos (dirCCW=1-> ascendente)
		);
	end component;
	component encoder_direction is
		port(
			clk_50M : in  std_logic;
			rst     : in  std_logic;
			sinA    : in  std_logic;
			sinB    : in  std_logic;
			dirCCW  : out std_logic;
			dirCW   : out std_logic
		);
	end component;
	signal sinA_c, sinB_c : std_logic;
	signal dirCCW, dirCW  : std_logic;
begin
antirreboteA : antirrebote
	generic map(
		--pulsos a 50MHz de duracion maxima del rebote
		D_MAX_REBOTE => 500 --10us duracion minima
	)
	port map(
		clk_50M => clk_50M,		--: in  std_logic;
		rst     => rst,			--: in  std_logic;
		sin     => sinA,		--: in  std_logic;
		sout    => sinA_c		--: out std_logic
	);
antirreboteB : antirrebote
	generic map(
		--pulsos a 50MHz de duracion maxima del rebote
		D_MAX_REBOTE => 500
	)
	port map(
		clk_50M => clk_50M,		--: in  std_logic;
		rst     => rst,			--: in  std_logic;
		sin     => sinB,		--: in  std_logic;
		sout    => sinB_c		--: out std_logic
	);
```

```
encoder_dir : encoder_direction
          port map(
                    clk_50M => clk_50M,             --: in  std_logic;
                    rst     => rst,                 --: in  std_logic;
                    sinA    => sinA_c,              --: in  std_logic;
                    sinB    => sinB_c,              --: in  std_logic;
                    dirCCW  => dirCCW,              --: out std_logic;
                    dirCW   => dirCW                --: out std_logic
          );

contador_frec : contador_frecuencia
          generic map(
                    --valor máximo 20
                    NUM_PULSOS_MEDIAR => 32         -- cuenta el número de pulsos de los que se van
                    --a hacer media para contar la frecuencia
          )
          port map(
                    clk_50M    => clk_50M,          --: in  std_logic;
                    rst        => rst,              --: in  std_logic;     --resetea el sistema, activo a nivel bajo
                    sinA       => sinA_c,           --: in  std_logic;
                    dirCCW     => dirCCW,           --: in  std_logic;
                    dirCW      => dirCW,            --: in  std_logic;
                    durPulsos  => durPulsos,        --: out std_logic_vector(31 downto 0); --duracion pulsos en 20ns
                    pulsos     => pulsos            --: out std_logic_vector(31 downto 0) -- suma el numero de pulsos
(dirCCW=1-> ascendente)
          );

                    CCW <= dirCCW;
                    CW  <= dirCW;

end architecture RTL;
```

Se realiza una simulación del bloque en una testbench, con ello se intenta simular el funcionamiento real de las señales.

En primer lugar, se generarán señales desfasadas, durante la primera fase la señal sinA irá adelantada a la señal sinB.

Como se puede ver en la simulación de la figura siguiente.

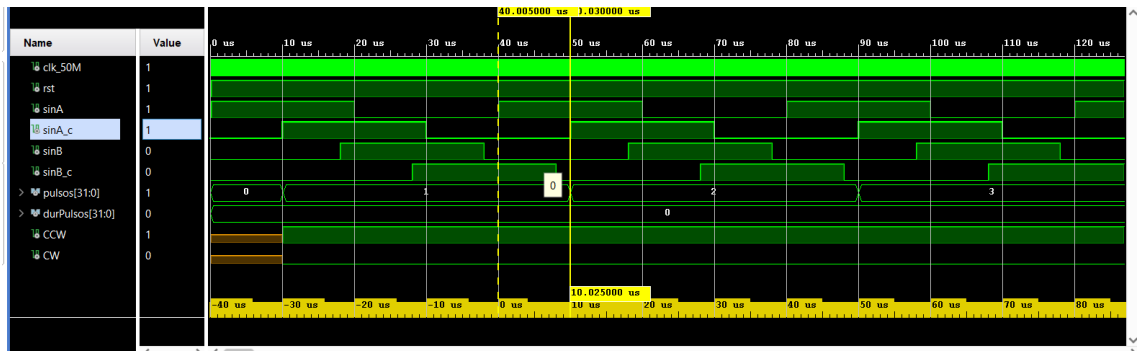


Figura 2-32 Simulación encoder_logic sim1

Como se puede observar en la siguiente captura, el valor de sinA se retrasa tantos microsegundos como sea la duración de pulso mínimo para que lo deje pasar el Antirrebote. En este caso son 10 us. Lo mismo sucederá en la señal sinB.

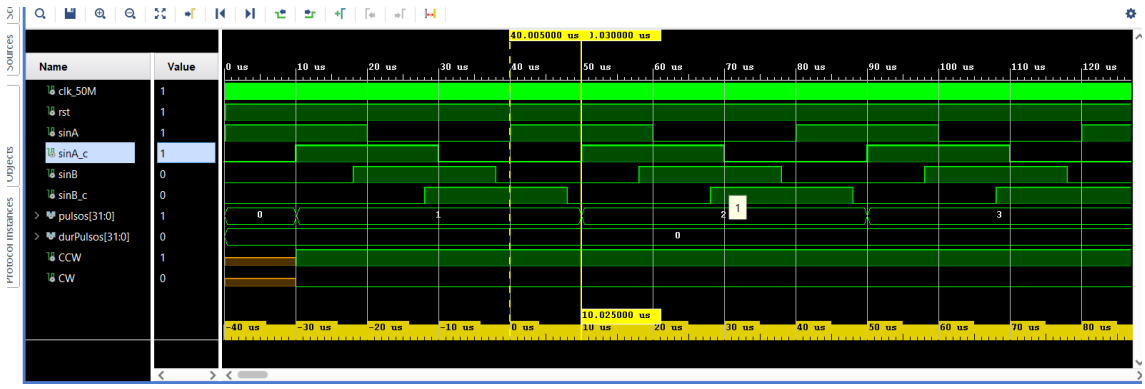


Figura 2-33 Simulación encoder_logic, retraso señal sinA

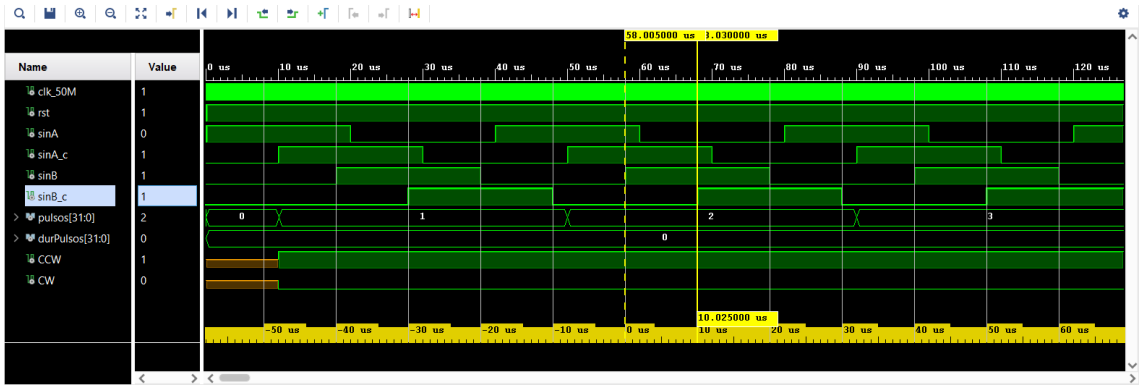


Figura 2-34 Simulación encoder_logic, retraso señal sinB

Por la forma en que se ha diseñado el Antirrebote, esto no afectará a la duración de la señal, y como se aplica a las dos señales el retraso se aplica a ambas señales.

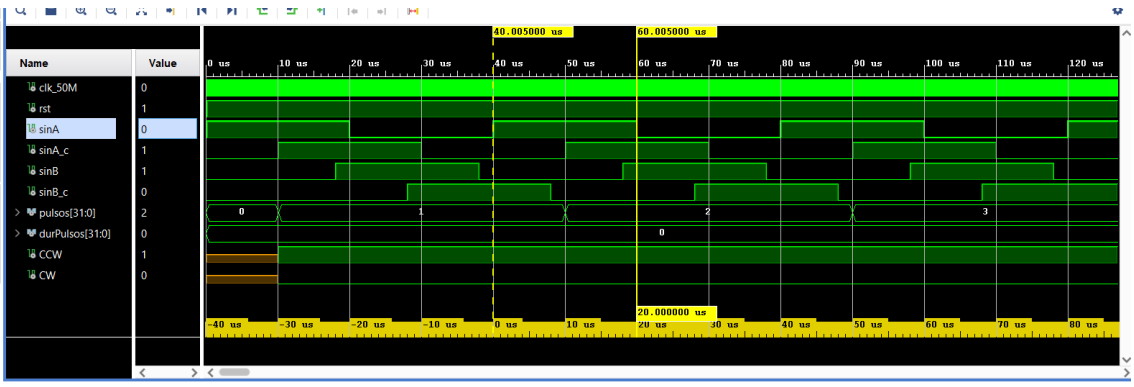


Figura 2-35 Simulación encoder_logic, duración es la misma en sinA que la de la salida del Antirrebote ,1.

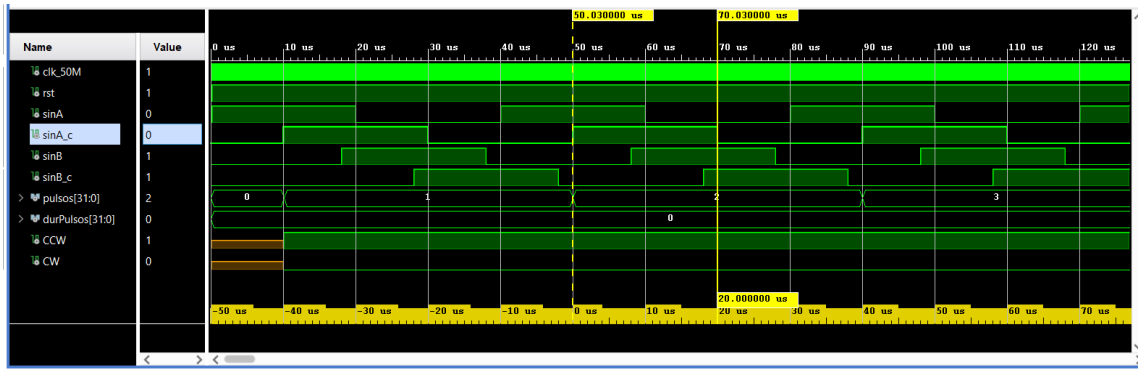


Figura 2-36 Simulación encoder_logic, duración es la misma en sinA que la de la salida del Antirrebote,1.

La dirección a la que está girando el eje del motor solo se actualiza cuando se detecta un flanco ascendente en la señal que sale del Antirrebote (sinA_c). El conteo de pulsos es además positivo.

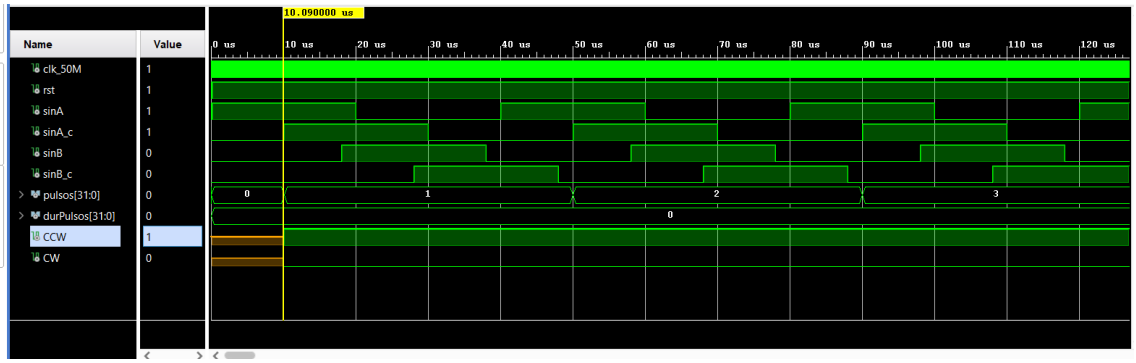


Figura 2-37 Simulación encoder_logic, funcionamiento de las señales de dirección CCW

Como se puede observar en la siguiente simulación, en cuanto se detecta que la señal sinA va retrasada respecto a la señal sinB, se cambia la dirección. Se puede comprobar también el funcionamiento de durPulsos. Son 32 pulsos que duran cada uno 20us, estos son 1000 pulsos.

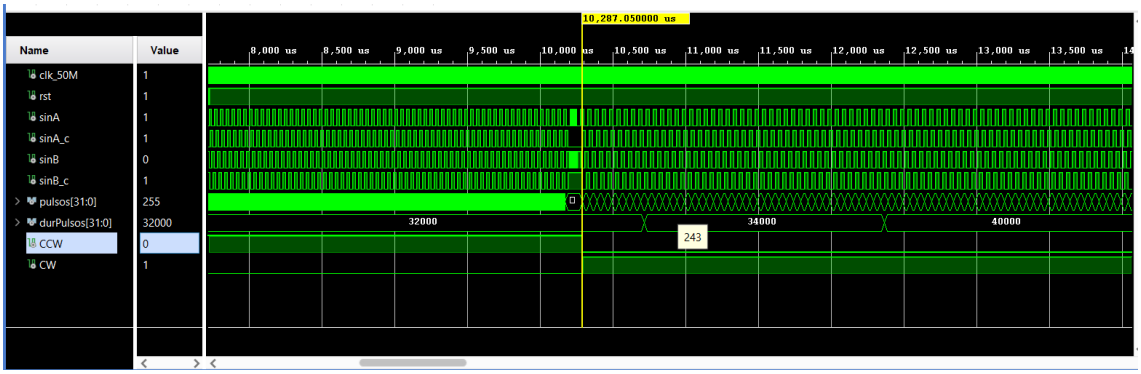


Figura 2-38 Simulación encoder_logic, funcionamiento de las señales de dirección CW

En la siguiente fase de comprobación se comprueba si deja pasar la señal que dura menos de 10us en sinA y sinB. Como se puede comprobar, no la deja pasar y además no cuenta. Los pulsos empiezan a decrecer y en cuanto la dirección se ha detectado que ha cambiado.

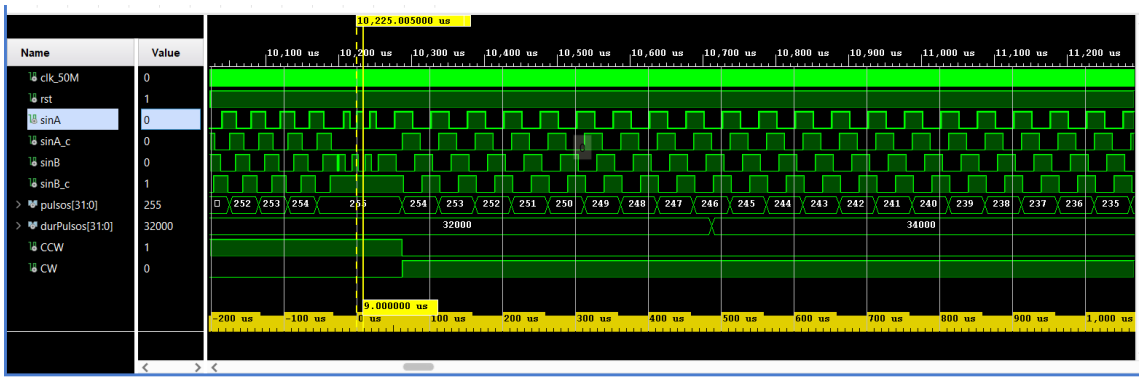


Figura 2-39 Simulación encoder_logic, funcionamiento del Antirrebote

El código de la testbench es el siguiente:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity testbench_encoder_logic is
end entity testbench_encoder_logic;

architecture RTL of testbench_encoder_logic is
	component encoder_logic is
		port(
			clk_50M    : in  std_logic;
			rst        : in  std_logic;
			sinA       : in  std_logic;
			sinB       : in  std_logic;
			rst_pulsos : in  std_logic;
			pulsos     : out std_logic_vector(31 downto 0);
			durPulsos  : out std_logic_vector(31 downto 0);
			CCW        : out std_logic;
			CW         : out std_logic
		);
	end component;
	signal clk_50M    : std_logic;
	signal rst        : std_logic;
	signal sinA       : std_logic;
	signal sinB       : std_logic;
	signal rst_pulsos : std_logic;
	signal pulsos     : std_logic_vector(31 downto 0);
	signal durPulsos  : std_logic_vector(31 downto 0);
	signal CCW        : std_logic;
	signal CW         : std_logic;

begin
	encoder_modulo : encoder_logic
		port map(
			clk_50M    => clk_50M,      --: in  std_logic;
			rst        => rst,          --: in  std_logic;
			sinA       => sinA,         --: in  std_logic;
			sinB       => sinB,         --: in  std_logic;
			rst_pulsos => rst_pulsos,   --: in  std_logic;
			pulsos     => pulsos,       --: out std_logic_vector(31 downto 0);
			durPulsos  => durPulsos,    --: out std_logic_vector(31 downto 0)
			CCW        => CCW,          --: out std_logic;
			CW         => CW            --: out std_logic
		);

	clock : process
	begin
		--50MHz clk
		clk_50M <= '0';
		wait for 10 ns;
		clk_50M <= '1';
		wait for 10 ns;
	end process;
	stimulus : process
	begin
		rst        <= '0', '1' after 50 ns;
		sinA       <= '0';
		sinB       <= '0';
		rst_pulsos <= '0';
		--rst_pulsos <= '1','0' after 50 ns;
		--retrasa las entradas para que este desincronizada del reloj
		wait for 5 ns;
		-- el módulo debe contar 255 pulsos y debe obtener una duracion
		--de 1000 en durPulso
		--la dirección es primero sinA, por tanto CCW='1' y CW='0'
		--es decir el motor estaría girando en sentido CCW
		primero_sinA : for I in 1 to 255 loop
			sinA <= '1';
			wait for 18 us;
			sinB <= '1';
			wait for 2 us;
			sinA <= '0';
			wait for 18 us;
			sinB <= '0';
			wait for 2 us;
		end loop;
		--las señales son de duración de 9us y por tanto no deverían ser detectadas

		primero_sinB : for I in 1 to 3 loop
			sinB <= '1';
			wait for 7 us;
			sinA <= '1';
			wait for 2 us;
			sinB <= '0';
			wait for 7 us;
			sinA <= '0';
			wait for 2 us;
		end loop;
```

```
                    --como la direccion ha cambiado, sinB antes de sinA, entonces
                    --el modulo debe detectar direccion CW='1' y CCW='0'
                    --además el contador de pulsos debe decrecer hasta -45
                    --la durPulso debe ser de 1250 porque la duracion del pulso es 25us
                    primero_sinB2 : for I in 1 to 300 loop
                              sinB <= '1';
                              wait for 23 us;
                              sinA <= '1';
                              wait for 2 us;
                              sinB <= '0';
                              wait for 23 us;
                              sinA <= '0';
                              wait for 2 us;
                    end loop;
                    rst_pulsos <= '1', '0' after 10 ns;
                    wait;
          end process;

end architecture RTL;
```

ANEXO III – Address Map Zynq-7000 SoC

La siguiente tabla ha sido obtenida de [4] y representa las direcciones físicas de las distintas memorias, así cómo las direcciones de los puertos GP0 y GP1, así como los registros de control del CPU y de los periféricos. La tabla también incluye quién puede acceder a esa dirección de memoria. Para poner un ejemplo de acceso ilegal, un puerto HP no puede acceder a los puertos MGP.

Address Range	CPUs and ACP	AXI_HP	Other Bus Masters(1)	Notes
0000_0000 to 0003_FFFF(2)	OCM	OCM	OCM	Address not filtered by SCU and OCM is mapped low
	DDR	OCM	OCM	Address filtered by SCU and OCM is mapped low
	DDR			Address filtered by SCU and OCM is not mapped low
				Address not filtered by SCU and OCM is not mapped low
0004_0000 to 0007_FFFF	DDR			Address filtered by SCU
				Address not filtered by SCU
0008_0000 to 000F_FFFF	DDR	DDR	DDR	Address filtered by SCU
		DDR	DDR	Address not filtered by SCU(3)
0010_0000 to 3FFF_FFFF	DDR	DDR	DDR	Accessible to all interconnect masters
4000_0000 to 7FFF_FFFF	PL		PL	General Purpose Port #0 to the PL, M_AXI_GP0
8000_0000 to BFFF_FFFF	PL		PL	General Purpose Port #1 to the PL, M_AXI_GP1
E000_0000 to E02F_FFFF	IOP		IOP	I/O Peripheral registers, see Table 4-6
E100_0000 to E5FF_FFFF	SMC		SMC	SMC Memories, see Table 4-5
F800_0000 to F800_0BFF	SLCR		SLCR	SLCR registers, see Table 4-3
F800_1000 to F880_FFFF	PS		PS	PS System registers, see Table 4-7
F890_0000 to F8F0_2FFF	CPU			CPU Private registers, see Table 4-4
FC00_0000 to FDFF_FFFF(4)	Quad-SPI		Quad-SPI	Quad-SPI linear address for linear mode
FFFC_0000 to FFFF_FFFF(2)	OCM	OCM	OCM	OCM is mapped high
				OCM is not mapped high

Tabla 2-4 Direcciones a nivel del sistema, direcciones físicas

ANEXO IV Partes:

ANEXO IV - I DC motor con caja reductora y dos encoder

Se han usado dos de estos motores.

El GB37 es un motor de corriente continua con las siguientes características:

Característica	Valor
Rated Voltage	12 V
Speed before deceleration	330 rpm
Idling current	250 ma
Power	4.8W
Blocking 122áximum current	6.5ª
Gearbox length	22 mm

Tabla 2-5 Características del motor

Figura 2-40 Motor de corriente continua GB37 con encoder

La reductora tiene un ratio de 1:30, es decir por cada 30 vueltas que da el eje del motor el eje de salida da 1. Los encoder detectan 13 pulsos por vuelta, pero lo que detectan son los giros del eje del motor y no del eje de salida.

Por lo tanto:

Característica	Valor
Pulsos por vuelta por encoder	390 pulsos
Grados por pulso	0.923 grados/pulso
Radianes por pulso	0.01611 rad/pulso

Tabla 2-6 Características de los pulsos de encoder

ANEXO IV - II AN10441: Level-shifter

Este módulo sirve para conectar señales de 5v a las entradas de la FPGA que funcionan a 3.3V.

Se usan en los encoders pues estos deben ser alimentados con 5V pues si se usan 3.3V la señal se ve muy afectada por el ruido y los niveles lógicos se ven distorsionados.

En GND se conecta el GND de la alimentación de la placa y en LV se alimenta 3.3V y en HV se alimenta 5V.

Todas las señales se colocan en los canales, HV1… HV4 y LV1… LV4. La hoja de característica dice que son intercambiables (las señales lógicas, pero no la alimentación) y cuando lo mides con el multímetro se puede observar que en los dos lados tienen el mismo voltaje.

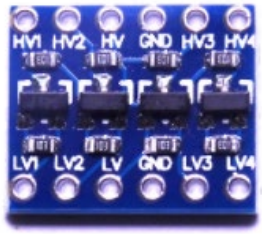


Figura 2-41 AN10441 Level-shifter

ANEXO IV - III TB6612FNG DC Motor Driver

Este driver es el encargado de permitir a la placa controlar los dos motores.

Es una opción optima puesto que permite juntar en un mismo módulo todas las señales necesarias para el control.

Este módulo será controlado por el periférico propio diseñado en la sección DISEÑO DRIVER-TB6612FNG.

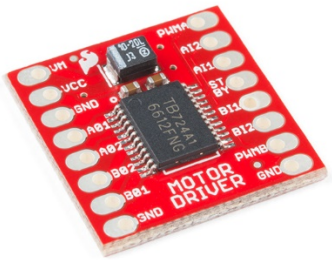


Figura 2-42 TB6612FNG Dual-Motor Driver

En la siguiente tabla se incluyen las funcionalidades de cada pin:

Nombre del PIN	Función	Poder/Entrada/Salida	
VM	Voltaje del motor	Poder	Aquí se provee potencia a los motores. (2.2V hasta 13.5V). En este caso, 7.5V.
VCC	Voltaje de la lógica	Poder	Se debe aplicar un voltaje soportado por el chip que va a comandar el módulo (2.7V hasta 5.5V). En el caso de la MiniZed, 3.3V.
GND	Neutro	Poder	Neutro común tanto de control como de potencia.
STBY	Habilita/Deshabilita el módulo	Entrada	Si STBY=1 el módulo puede trabajar, sino se le da una señal HIGH, es tirado a 0 por una resistencia PULL-DOWM.
AIN1/AIN2	Dirección o freno del motor A	Entrada	Determina la Dirección Si AIN1=0 y AIN2=1 Dirección= CCW Si AIN1=AIN2 Freno por cortocircuito de los devanados.
BIN1/BIN2	Dirección o freno del motor B	Entrada	Determina la Dirección Si BIN1=0 y BIN2=1 Dirección= CCW Si BIN1=BIN2 Freno por cortocircuito de los devanados.
PWMA/PWMB	Entrada PWM para el control de A/B	Entrada	La señal PWM controla la velocidad del motor. La frecuencia de la señal no puede ser mayor de 100K.
A01/B01	Salidas a los dos motores.	Salida	Se conectan a los motores y si se conecta A01 a M+ del motor, entonces el motor seguirá lo especificado en la dirección.

Tabla 2-7 Pines y funcionamiento TB6612FNG

Sus características eléctricas son:

Característica	Valor
Alimentación	15v Máx
Corriente de salida	1.2 A (media) y 3.2 A (pico)
Alimentación del circuito de control	2.7-5V

Tabla 2-8 Características eléctricas del driverq

ANEXO IV - IV MiniZed

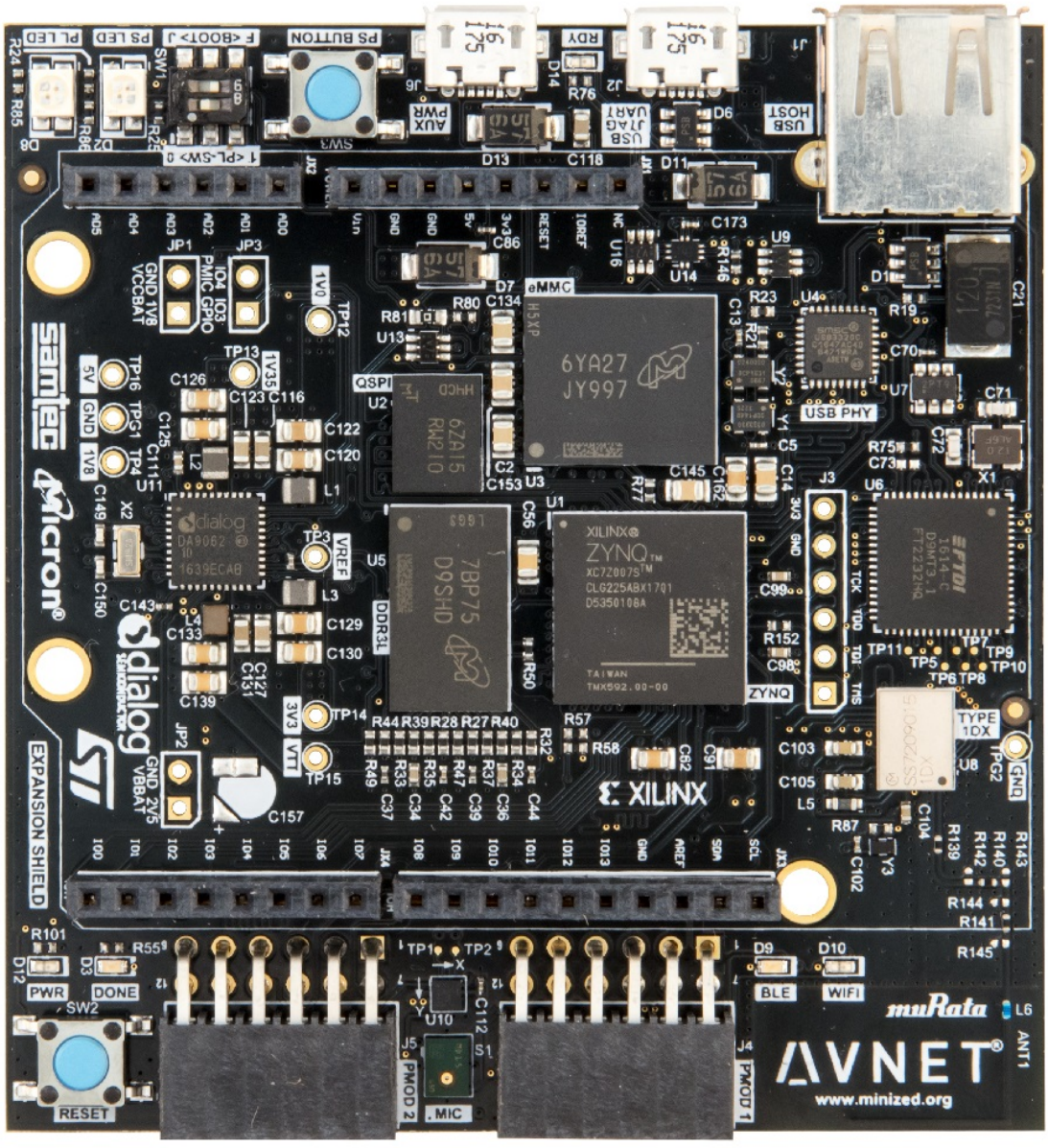


Figura 2-43 MiniZed vista frontal

El diagrama de bloques de esta placa es el siguiente:

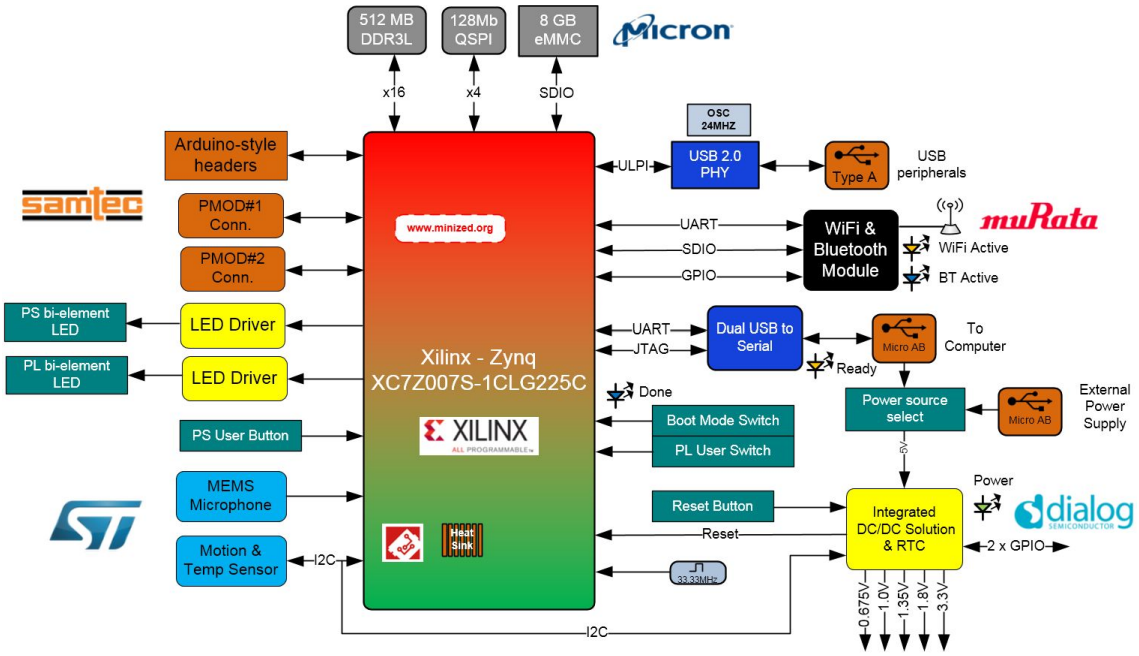


Figura 2-44 Diagrama de bloques de la MiniZed

En los Planos 2 – 7 del documento PLANOS se presentan los esquemas de conexiones de la placa.

ANEXO IV - V Regulador Lineal L7805CV

Una imagen del regulador se puede apreciar en la siguiente figura.

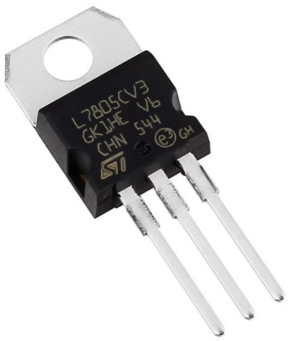


Figura 2-45 L7805CV

Un esquema de conexión típico es el siguiente:

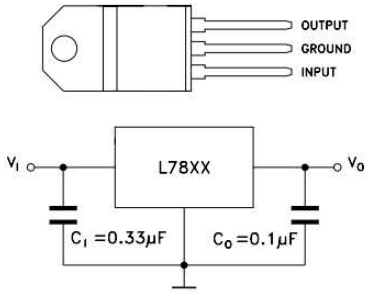


Figura 2-46 Esquema de conexión L7805CV

ANEXO IV - VI MPU 6050

Este dispositivo cuenta con un acelerómetro y un giroscopio. Se usará para conocer la inclinación del robot.

Cuenta con un protocolo de comunicación I2C.

El esquema de sus conexiones está en el Plano 11 del documento Planos.

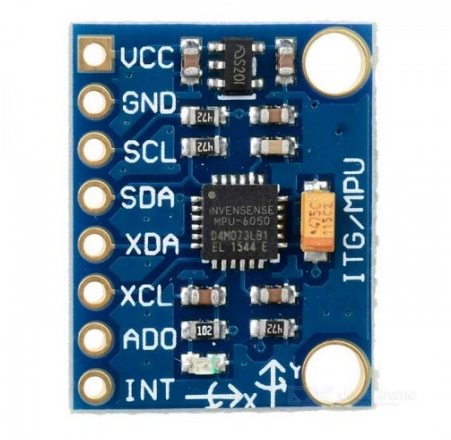


Figura 2-47 MPU6050

Se ha diseñado una librería para obtener los valores de los registros.

ANEXO V Código Matlab

Comprobación cálculo función de transferencia $\boldsymbol{\theta(s)/V(s)}$

```
syms s w_1 w_2  R K r phi_dot V theta m_p m_r l g I_r I_p
```

Despejando la phi_dot ecuacion 1-58

```
phi_dot=(s^2*(I_p+m_p*l^2)+m_p*l*g)/(s*(-m_p*l*r))*theta
```

```
phi_dot =
```

$$-\frac{\theta\left(\left(m_p l^2+I_p\right) s^2+g\, l\, m_p\right)}{l\, m_p\, r\, s}$$

Despejando de la ecuacion 1-56 V

```
V=(phi_dot*(s*(2*m_r*r^2-m_p*r^2-2*I_r)*R+30^2*K^2)/(r*R)-
theta*s^2*m_p*l)*r*R/(60*K)
```

```
V =
```

$$-\frac{R\, r\left(l\, m_p\, s^2\,\theta+\frac{\theta\left(\left(m_p l^2+I_p\right) s^2+g\, l\, m_p\right)\left(900\,K^2-R\, s\left(2\, I_r+m_p\, r^2-2\, m_r\, r^2\right)\right)}{R\, l\, m_p\, r^2\, s}\right)}{60\, K}$$

```
FT=simplifyFraction(theta/V)
```

```
FT =
```

$$-\frac{60\, K\, l\, m_p\, r\, s}{900\,K^2\, l^2\, m_p\, s^2+900\, g\, K^2\, l\, m_p+900\, I_p\, K^2\, s^2+2\, R\, m_r\, l^2\, m_p\, r^2\, s^3-2\, I_r\, R\, l^2\, m_p\, s^3-R\, g\, l\, m_p^2\, r^2\, s+2\, R\, g\, m_r\, l\, m_p\, r^2\, s-2\, I_r\, R\, g\, l\, m_p\, s-I_p\, R\, m_p\, r^2\, s^3+2\, I_p\, R\, m_r\, r^2\, s^3-2\, I_p\, I_r\, R\, s^3}$$

```
syms w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8
syms s
syms R K r phi_dot V theta m_p m_r l g I_r I_p
w_1=2*m_r*r^2-m_p*r^2-2*I_r;
w_2=I_p+m_p*l^2;
w_3=m_p*l*g;
w_4=-m_p*l*r;
w_5=w_1*R;
w_6=30^2*K^2;
w_7=-60*K;
w_8=m_p*l;
FT_comp=-s*w_4*w_7/(s^3*(w_2*w_5-
w_4*w_8*r*R)+s^2*w_2*w_6+s*w_3*w_5+w_3*w_6)
```

```
FT_comp =
```

$$\frac{60\, K\, l\, m_p\, r\, s}{s^3\left(R\left(m_p l^2+I_p\right)\sigma_1-R\, l^2\, m_p^2\, r^2\right)-900\, K^2\, s^2\left(m_p l^2+I_p\right)-900\, K^2\, g\, l\, m_p+R\, g\, l\, m_p\, s\,\sigma_1}$$

where

$$\sigma_1 = 2\, I_r + m_p\, r^2 - 2\, m_r\, r^2$$

```
FT_comp=expand(FT_comp)
```

```
FT_comp =
```

$$-\frac{60\, K\, l\, m_p\, r\, s}{900\,K^2\, l^2\, m_p\, s^2+900\, g\, K^2\, l\, m_p+900\, I_p\, K^2\, s^2+2\, R\, m_r\, l^2\, m_p\, r^2\, s^3-2\, I_r\, R\, l^2\, m_p\, s^3-R\, g\, l\, m_p^2\, r^2\, s+2\, R\, g\, m_r\, l\, m_p\, r^2\, s-2\, I_r\, R\, g\, l\, m_p\, s-I_p\, R\, m_p\, r^2\, s^3+2\, I_p\, R\, m_r\, r^2\, s^3-2\, I_p\, I_r\, R\, s^3}$$

Comprobación que son iguales:

```
(FT-FT_comp)
```

```
ans =
0
```

Código empleado para leer la velocidad del robot para encontrar K.

```
#include <stdio.h>
#include "xparameters.h"
#include "xscutimer.h"
#include "xscugic.h"
#include "xil_exception.h"
#include "xil_printf.h"
#include "sleep.h"
#include <stdbool.h>
//módulos propios
#include "driver_tb6612fng.h"
#include "encoder_motor.h"

#define PI 3.14159265358979323846
#define durPulso50M 20E-9
//Identificacion del modulo SCU Private Timer
#define TIMER_DEVICE_ID                 XPAR_XSCUTIMER_0_DEVICE_ID
//GenericInterruptController Identificacion 0
#define INTC_DEVICE_ID                  XPAR_SCUGIC_SINGLE_DEVICE_ID
//Numero de indentificacion de la interrupcion
#define TIMER_IRPT_INTR                 XPAR_SCUTIMER_INTR

#define TIMER_LOAD_VALUE        0x1FCA052
int num_rep=0;
u32 driver_add=XPAR_DRIVER_TB6612FNG_0_S00_AXI_BASEADDR;
u32 dirEncoder=XPAR_ENCODER_MOTOR_0_S00_AXI_BASEADDR;
XScuTimer TimerInstance;        /* Cortex A9 Scu Private Timer Instance */
XScuGic IntcInstance;           /* Interrupt Controller Instance */

void InicializaTimer(u16 TimerDeviceId,XScuTimer *TimerInstancePtr){
          XScuTimer_Config *ConfigPtr;
          ConfigPtr = XScuTimer_LookupConfig(TimerDeviceId);
          XScuTimer_CfgInitialize(TimerInstancePtr, ConfigPtr,ConfigPtr->BaseAddr);
}
void vel_rueda1(float time,float* rpm, float* rads){
          int32_t pulsos;
          float frec;
          pulsos=ENCODER_MOTOR_mReadReg(dirEncoder,ENCODER_MOTOR_S00_AXI_SLV_REG1_OFFSET);
          frec=pulsos/time;
           *rpm=60.0*frec/(30.0*13.0);
           *rads=(*rpm)*2.0*PI/60.0;//rad/s
}
void vel_rueda2(float time,float* rpm, float* rads){
          int32_t pulsos;
          float frec;
          pulsos=ENCODER_MOTOR_mReadReg(dirEncoder,ENCODER_MOTOR_S00_AXI_SLV_REG5_OFFSET);
          frec=pulsos/time;
           *rpm=60.0*frec/(30.0*13.0);
           *rads=(*rpm)*2.0*PI/60.0;//rad/s
}

void HandlerTimer(void *CallBackRef){
          static float time=0.1;
          static float rpm,rads;
          static u32 pulsos;
          XScuTimer *TimerInstancePtr = (XScuTimer *) CallBackRef;
          XScuTimer_ClearInterruptStatus(TimerInstancePtr);
          num_rep++;
          printf("Han pasado segundos %f desde el inicio:\r\n",num_rep/1000.0);
          vel_rueda1(time,&rpm,&rads);
          write_encoder_motor_en_1(2);
          printf("Este es el valor de la velocidad en la rueda derecha:\r\n rpm %.6f y rad/s %6.6f\r\n",rpm,rads);
          printf("\x1B[3A");//VT100 escape code:move cursor up 9 lines
}
void HabilitaSCUTimer(XScuGic *IntcInstancePtr, XScuTimer *TimerInstancePtr,u16 TimerIntrId){
                    XScuGic_Config *IntcConfig;
                    //Mira la configuracion del GIC
                    IntcConfig = XScuGic_LookupConfig(INTC_DEVICE_ID);
                    //Configura el SCU PT
                    XScuGic_CfgInitialize(IntcInstancePtr, IntcConfig,IntcConfig->CpuBaseAddress);
                    Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_IRQ_INT,(Xil_ExceptionHandler)XScuGic_InterruptHandler,
                                                 IntcInstancePtr);
                    //Conecta la interrupcion del SCU Private Timer con el HandlerTimer
                    XScuGic_Connect(IntcInstancePtr, TimerIntrId,(Xil_ExceptionHandler)HandlerTimer,
                                    (void *)TimerInstancePtr);
                    //Habilita el GIC
                    XScuGic_Enable(IntcInstancePtr, TimerIntrId);
                    //Habilita la interrupcion del SCU PrivateTimer
                    XScuTimer_EnableInterrupt(TimerInstancePtr);
                    //Habilita las interrupciones del micro
                    Xil_ExceptionEnable();

}
```

```
int main(void)
{
	//Inicializa el módulo SCU Private Timer
	InicializaTimer(TIMER_DEVICE_ID,&TimerInstance);
	//Habilita la interrupion del SCU Private Timer
	HabilitaSCUTimer(&IntcInstance,&TimerInstance,TIMER_IRPT_INTR);
	XScuTimer_EnableAutoReload(&TimerInstance);
	XScuTimer_LoadTimer(&TimerInstance, TIMER_LOAD_VALUE);
	XScuTimer_Start(&TimerInstance);

	write_encoder_motor_en_1(1);
	write_encoder_motor_en_2(1);


	while(1){


	}
	return 0;
}
```

ANEXO VI SOLUCIÓN FINAL

El esquema eléctrico de la solución está en el Plano 11 del documento de Planos.

ANEXO VI - I Código de la Interrupción del SCU Private Timer

```
//Identificacion del modulo SCU Private Timer
#define TIMER_DEVICE_ID                 XPAR_XSCUTIMER_0_DEVICE_ID
//GenericInterruptController Identificacion 0
#define INTC_DEVICE_ID                  XPAR_SCUGIC_SINGLE_DEVICE_ID
//Numero de indentificacion de la interrupcion
#define TIMER_IRPT_INTR                 XPAR_SCUTIMER_INTR
//Valor que se va a cargar en el contador, que está habilitado en cuenta descendente
//Hay que tener en cuenta que el SCU Private timer funciona a la mitad de la velocidad del CLK del CPU
#define TIMER_LOAD_VALUE        0x1FCA052


int num_rep=0;
//Valores de la direccion base de los módulos driver_tb6612fng y encoder_motor
u32 driver_add=XPAR_DRIVER_TB6612FNG_0_S00_AXI_BASEADDR;
u32 dirEncoder=XPAR_ENCODER_MOTOR_0_S00_AXI_BASEADDR;
//Estructuras que usan las funciones de xilinx
XScuTimer TimerInstance;        /* Cortex A9 Scu Private Timer Instance */
XScuGic IntcInstance;           /* Interrupt Controller Instance */

void InicializaTimer(u16 TimerDeviceId,XScuTimer *TimerInstancePtr){
          XScuTimer_Config *ConfigPtr;
          ConfigPtr = XScuTimer_LookupConfig(TimerDeviceId);
          XScuTimer_CfgInitialize(TimerInstancePtr, ConfigPtr,ConfigPtr->BaseAddr);
}
//Manejador donde poner lo que sucede cuando pasa la interrupcion
void HandlerTimer(void *CallBackRef){
          XScuTimer *TimerInstancePtr = (XScuTimer *) CallBackRef;
//Limpia la interrupcion
          XScuTimer_ClearInterruptStatus(TimerInstancePtr);
}
void HabilitaSCUTimer(XScuGic *IntcInstancePtr, XScuTimer *TimerInstancePtr,u16 TimerIntrId){
                    XScuGic_Config *IntcConfig;
                    //Mira la configuracion del GIC
                    IntcConfig = XScuGic_LookupConfig(INTC_DEVICE_ID);
                    //Configura el SCU PT
                    XScuGic_CfgInitialize(IntcInstancePtr, IntcConfig,IntcConfig->CpuBaseAddress);
                    Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_IRQ_INT,(Xil_ExceptionHandler)XScuGic_InterruptHandler,
                                                 IntcInstancePtr);
                    //Conecta la interrupcion del SCU Private Timer con el HandlerTimer
                    XScuGic_Connect(IntcInstancePtr, TimerIntrId,(Xil_ExceptionHandler)HandlerTimer,
                                                (void *)TimerInstancePtr);
                    //Habilita el GIC
                    XScuGic_Enable(IntcInstancePtr, TimerIntrId);
                    //Habilita la interrupcion del SCU PrivateTimer
                    XScuTimer_EnableInterrupt(TimerInstancePtr);
                    //Habilita las interrupciones del micro
                    Xil_ExceptionEnable();

}


int main(void)
{
          //Inicializa el módulo SCU Private Timer
          InicializaTimer(TIMER_DEVICE_ID,&TimerInstance);
          //Habilita la interrupion del Private Timer
          HabilitaSCUTimer(&IntcInstance,&TimerInstance,TIMER_IRPT_INTR);
          //Configuracion del funcionamiento del timer
          XScuTimer_EnableAutoReload(&TimerInstance);
          XScuTimer_LoadTimer(&TimerInstance, TIMER_LOAD_VALUE);
          XScuTimer_Start(&TimerInstance);
}
```

ANEXO VI - II Funciones para el manejo del MPU6050

```
/*
 * mpu_6050.h
 *
 *  Created on: 17 ago. 2020
 *      Author: roosb
 */

#ifndef SRC_MPU_6050_H_
#include <stdio.h>
#include <xbasic_types.h>
#include "xparameters.h"
#include "xscutimer.h"
#include "xscugic.h"
//módulo I2C0 del PS
#include "xiicps.h"

#define SRC_MPU_6050_H_

#define MPU6050_RW_SMPLRT_DIV                   0x19   //divisor

#define MPU6050_RW_GYRO_CONFIG                  0x1B //[4:3] FS_SEL
#define MPU6050_RW_ACCEL_CONFIG     0x1C//[4:3] AFS_SEL

#define MPU6050_RW_INT_PIN_CFG                  0x37//[int_level int_open latch_int_en int_rd_clear fsync_int_en i2c_bypass_en - ]
//en reinicio 0x00 Izq a derecha Int pin active high, push pull conf, Int pin emits 50us pulse, interrupt_status cleared by reading int_status(reg58)
//igual combiene modificar bit 4 y ponerlo a 1 para que con cualquier lectura se clear flag
#define MPU6050_RW_INT_ENABLE                   0x38//bit 0 data ready enable

#define MPU6050_RA_ACCEL_XOUT_H    0x3B
#define MPU6050_RA_ACCEL_XOUT_L    0x3C
#define MPU6050_RA_ACCEL_YOUT_H    0x3D
#define MPU6050_RA_ACCEL_YOUT_L    0x3E
#define MPU6050_RA_ACCEL_ZOUT_H    0x3F
#define MPU6050_RA_ACCEL_ZOUT_L    0x40
#define MPU6050_RA_TEMP_OUT_H      0x41
#define MPU6050_RA_TEMP_OUT_L      0x42
#define MPU6050_RA_GYRO_XOUT_H     0x43
#define MPU6050_RA_GYRO_XOUT_L     0x44
#define MPU6050_RA_GYRO_YOUT_H     0x45
#define MPU6050_RA_GYRO_YOUT_L     0x46
#define MPU6050_RA_GYRO_ZOUT_H     0x47
#define MPU6050_RA_GYRO_ZOUT_L     0x48
#define MPU6050_RA_WHOIAM                       0x75
```

```
#define MPU6050_RW_PWR_MGMT_1                       0x6B
#define MPU6050_RW_SIGNAL_PATH_RESET 0x68 //[2:0] gyroReset accelReset tempReset
/*
 *Controla el tamaño del los bufferes i2c
 */

#define BUFFER_SIZE          132
/***************Para el i2c MPU 6050*********************/
/*
 * The slave address to send to and receive from.
 */
#define IIC_SLAVE_ADDR                      0x68
#define IIC_SCLK_RATE                       100000
#define IIC_DEVICE_ID                       XPAR_XIICPS_0_DEVICE_ID



 extern long gyro_x_offset;
 extern long gyro_y_offset;
 extern long gyro_z_offset;
extern int16_t accel_x,accel_y,accel_z,giro_x,giro_y,giro_z;
extern float temp;
extern u8 SendBuffer[BUFFER_SIZE];    /**< Buffer for Transmitting Data */
extern u8 RecvBuffer[BUFFER_SIZE];    /**< Buffer for Receiving Data */


int receiveToMPU6050(XIicPs* mpu6050_i2c,u32 dirReg,u8* data,u8 len);
int writeToMPU6050(XIicPs* mpu6050_i2c, u32 dirReg,u8* valor);
void mpu_ReadSensors(XIicPs* mpu6050_i2c);
int initMPU6050IIC(XIicPs* mpu6050_i2c,XIicPs_Config *Config);
void mpu_printfAll();
#endif /* SRC_MPU_6050_H_ */
```

```
#include "mpu_6050.h"

int receiveToMPU6050(XIicPs* mpu6050_i2c,u32 dirReg,u8* data,u8 len){

            int Status;

SendBuffer[0]=dirReg;
           // printf("Transmitiendo datos %d\n",dirReg);
            //Escribe la direccion de la que se va a leer
                     Status = XIicPs_MasterSendPolled(mpu6050_i2c, SendBuffer,
                                                1, IIC_SLAVE_ADDR);
                     if (Status != XST_SUCCESS) {
                               printf("Error de transmision %d\n",dirReg);
                               return XST_FAILURE;
                     }
           //        printf("Esperando a que termine... %d \n",dirReg);
                     /*
                      * Wait until bus is idle to start another transfer.
                      */
                     while (XIicPs_BusIsBusy(mpu6050_i2c)) {
                               /* NOP */
                     }
                     Status = XIicPs_MasterRecvPolled(mpu6050_i2c, data,
                                                      len, IIC_SLAVE_ADDR);
                               if (Status != XST_SUCCESS) {
                                         printf("Error de recepcion de la direccion: %d\n",dirReg);
                                         return XST_FAILURE;
                               }


//          printf("Transmision completa %d\n",dirReg);



           return XST_SUCCESS;
}
int writeToMPU6050(XIicPs* mpu6050_i2c, u32 dirReg,u8* valor){
 u8 buffer[2];
 int Status;

 // Fill the buffer with required data
 SendBuffer[0] =dirReg;
 SendBuffer[1]=*valor;
           Status = XIicPs_MasterSendPolled(mpu6050_i2c, SendBuffer,
                                2, IIC_SLAVE_ADDR);
           if (Status != XST_SUCCESS) {

           printf("Error de transmision %d\n",*valor);
                     return XST_FAILURE;
           }
           // printf("Transmitiendo datos %d\n",*valor);
            //
           /*
            * Wait until bus is idle to start another transfer.
            */
           while (XIicPs_BusIsBusy(&mpu6050_i2c)) {
                     /* NOP */
           }


            //printf("Transmision completa %d\n",*valor);
            return XST_SUCCESS;
           }
           void mpu_ReadSensors(XIicPs* mpu6050_i2c){
                     int i;
                     u8 read_data[14];
                     int16_t datos_unidos[7];
                     receiveToMPU6050(mpu6050_i2c,MPU6050_RA_ACCEL_XOUT_H,&read_data,14);
                     //transformacion de datos leidos a valores reales

                                         datos_unidos[0]=read_data[0]<<8|read_data[1];//
                                         datos_unidos[1]=read_data[2]<<8|read_data[3];//
                                         datos_unidos[2]=read_data[4]<<8|read_data[5];//
                                         datos_unidos[3]=read_data[6]<<8|read_data[7];//
                                         datos_unidos[4]=read_data[8]<<8|read_data[9];//
                                         datos_unidos[5]=read_data[10]<<8|read_data[11];//
                                         datos_unidos[6]=read_data[12]<<8|read_data[13];//
                               //Aceleraciones
                                         accel_x=datos_unidos[0];
                                         accel_y=datos_unidos[1];
                                         accel_z=datos_unidos[2];


                               //Datos de la temperatura
                                         temp=datos_unidos[3]/340.0+36.53;


                                         //Los datos del gyro quitandole el offset que tiene
                                         giro_x=datos_unidos[4]-gyro_x_offset;
                                         giro_y=datos_unidos[5]-gyro_y_offset;
                                         giro_z=datos_unidos[6]-gyro_z_offset;
           }
```

```
int initMPU6050IIC(XIicPs* mpu6050_i2c,XIicPs_Config *Config){
 int Status;

int Index;


 u8 reg;
 float datos[7];
 int len,samples;
 long gyro_x_offset_aux=0;
 long gyro_y_offset_aux=0;
 long gyro_z_offset_aux=0;

 Config = XIicPs_LookupConfig(IIC_DEVICE_ID);
          if (NULL == Config) {
                    return XST_FAILURE;
          }

          Status = XIicPs_CfgInitialize(mpu6050_i2c, Config, Config->BaseAddress);
          if (Status != XST_SUCCESS) {
                    return XST_FAILURE;
          }

          /*
                     * Set the IIC serial clock rate.
                     */
                    XIicPs_SetSClk(mpu6050_i2c, IIC_SCLK_RATE);
                    /*
                              Inicializar los valores de los buffer
                               */
                              for (Index = 0; Index < BUFFER_SIZE; Index++) {
                                        SendBuffer[Index] = 0;
                                        RecvBuffer[Index] = 0;
                              }


                    //Sacar de sleep a MPU6050 (al reset está a SLEEP=1) y configura como refrencia del clk el X axis
                    reg=1U;
                    writeToMPU6050(mpu6050_i2c,MPU6050_RW_PWR_MGMT_1,&reg);
                    //Configura el sample rate del GYRO de 8KH del reset a 1KHz para que sea de 1KHz
                    reg=7U;
                    writeToMPU6050(mpu6050_i2c,MPU6050_RW_SMPLRT_DIV,&reg);
                    //Configuracion de la escala del acc a 4g
                    reg=0x08;
                    writeToMPU6050(mpu6050_i2c,MPU6050_RW_ACCEL_CONFIG,&reg);
                    //Configuracion de la escala del gyro a 2000
                    reg=0x18;
                    writeToMPU6050(mpu6050_i2c,MPU6050_RW_GYRO_CONFIG,&reg);

                    //Cálculo del offset del giroscopio
                    for(int cycle_count = 0; cycle_count < 20; cycle_count++){    //This loop takes 2000 readings of the
gyro x,y, and z axes
                    mpu_ReadSensors(mpu6050_i2c);                     //Grab raw MPU6050 values
                    gyro_x_offset_aux += giro_x;              //Add new raw value to total gyro value
                    gyro_y_offset_aux += giro_y;
                    gyro_z_offset_aux += giro_z;
                      usleep(4000);                //add 3us delay to simulate 250hz loop.
                    }
                    gyro_x_offset_aux /= 20;   //divide by 2000 to get average calibration values
                    gyro_y_offset_aux /= 20;
                    gyro_z_offset_aux /= 20;
                    gyro_x_offset=gyro_x_offset_aux;
                    gyro_y_offset=gyro_y_offset_aux;
                    gyro_z_offset=gyro_z_offset_aux;



 return XST_SUCCESS;

}

void mpu_printfAll(){
          //se supone que vamos a conservar el
          //3 de acc 1 temp y 3 de gyro
          int i;

                                        printf("Valor de la acceleracion eje x es %.2f\r\n",accel_x/8192.0);
                                        printf("Valor de la acceleracion eje y es %.2f\r\n",accel_y/8192.0);
                                        printf("Valor de la acceleracion eje z es %.2f\r\n",accel_z/8192.0);


                                        printf("Valor de la temperatura es %.2f\r\n",temp);


                                        printf("Valor del Gyro eje x es %.2f\r\n",giro_x/16.4);
                                        printf("Valor del Gyro eje y es %.2f\r\n",giro_y/16.4);
                                        printf("Valor del Gyro eje z es %.2f\r\n",giro_z/16.4);
                    //        printf("\x1B[7A");//VT100 escape code:move cursor up 7 lines
}
```

ANEXO VI - III Código del programa final

```
#include <stdio.h>
#include "xparameters.h"
#include "xscutimer.h"
#include "xscugic.h"
#include "xil_exception.h"
#include "xil_printf.h"
#include "sleep.h"
#include <stdbool.h>
#include <math.h>
#include <stdbool.h>
//módulo I2C0 del PS
#include "xiicps.h"
//módulos propios
#include "driver_tb6612fng.h"
#include "encoder_motor.h"
#include "mpu_6050.h"



 /*
  * The following buffers are used in this example to send and receive data
  * with the IIC.
  */
 u8 SendBuffer[BUFFER_SIZE];    /**< Buffer for Transmitting Data */
 u8 RecvBuffer[BUFFER_SIZE];    /**< Buffer for Receiving Data */


 long gyro_x_offset=0;
 long gyro_y_offset=0;
 long gyro_z_offset=0;
 int16_t accel_x,accel_y,accel_z,giro_x,giro_y,giro_z;
 float temp;

 /************Instancias para el i2CPS************/
 XIicPs mpu6050_i2c;            /**< Instance of the IIC Device */
 XIicPs_Config* Config;

//***************Para el filtro*******************/
#define PI 3.14159265358979323846
#define durPulso50M 20E-9
//Identificacion del modulo SCU Private Timer
#define TIMER_DEVICE_ID                 XPAR_XSCUTIMER_0_DEVICE_ID
//GenericInterruptController Identificacion 0
#define INTC_DEVICE_ID                  XPAR_SCUGIC_SINGLE_DEVICE_ID
//Numero de indentificacion de la interrupcion
#define TIMER_IRPT_INTR                 XPAR_SCUTIMER_INTR
//Repeticion cada 10ms
#define TIMER_LOAD_VALUE        0x32DCD2
int num_rep=0;
u32 driver_add=XPAR_DRIVER_TB6612FNG_0_S00_AXI_BASEADDR;
u32 dirEncoder=XPAR_ENCODER_MOTOR_0_S00_AXI_BASEADDR;
XScuTimer TimerInstance;        /* Cortex A9 Scu Private Timer Instance */
XScuGic IntcInstance;           /* Interrupt Controller Instance */

/*******************Variables control*********************************/
float referencia=0;
float T_muestreo=0.01;
float kp = 150;
float ki = 200;
float kd = .5;
/*******************Funciones de control*********************************/
void PID (float inclinacion,float referencia,float* accion_control,bool* motor_on){
          float e_k,actuacion_proporcional,actuacion_integral,regulacion_derivativa;
          static float Error_integral=0;
          static float e_k_anterior=0;
          //error en la referencia
          e_k = referencia - inclinacion;
          if (e_k>40){
                    *motor_on=false;
                    Error_integral=0;
          }
          else if (e_k<-40){
                    *motor_on=false;
                    Error_integral=0;
          }
          else{
          //acumulacion del error integral
          Error_integral = Error_integral + e_k * T_muestreo;
          actuacion_proporcional=kp*e_k;
          actuacion_integral=ki*Error_integral;
          regulacion_derivativa=kd*(e_k-e_k_anterior)/T_muestreo;

          printf("Valor de error %.2f\r\n",actuacion_proporcional);
          printf("Valor de Ierror %.2f\r\n",actuacion_integral);
          printf("Valor de dErr %.2f\r\n",regulacion_derivativa);
          *accion_control=actuacion_proporcional+actuacion_integral+regulacion_derivativa;
          *motor_on=true;
          printf("El valor de la accion de control es %.2f, motor encedido: %d\r\n",*accion_control,*motor_on);
          e_k_anterior=e_k;
          }
}
```

```
void control_accion(float accion_control,bool motor_on){

          if (accion_control > 255)
          accion_control = 255;
          else if (accion_control < -255)
          accion_control = -255;
          if (motor_on==false){
                    driver_motor_timA(0);
                    driver_motor_timB(0);
          }
          else if (accion_control > 0)
                   {
                      driver_motor_dir(1,1);
                      driver_motor_timA(accion_control);
                      driver_motor_timB(accion_control);

                   }
          else  if (accion_control < 0)
                   {
                     driver_motor_dir(0,0);
                     accion_control=-accion_control;
                     driver_motor_dir(1,1);
                     driver_motor_timA(accion_control);
                     driver_motor_timB(accion_control);
                   }

}
/********************Fin funciones control**************************/
/************Funciones del MPU6050**********/


void FiltroComplementario(float* inclinacion){

float inclinacion_acc,inclinacion_giro;
//8192 es el valor del lsb de los datos del accelerómetro
          inclinacion_acc = atan(-1*(accel_x/8192.0)/sqrt(pow((accel_y/8192.0),2) + pow((accel_z/8192.0),2)));
          //Conversión a radianes
          inclinacion_acc=inclinacion_acc*180/PI;


          //Se aplica el Filtro Complementario
          //16.4 es el valor del lsb de las medidas del giro
          inclinacion_giro=giro_y/16.4 * 0.01;
          //Los valores de las ponderaciones filtro pasabajos y el filtro pasaaltos debe sumar 1
           *inclinacion = 0.97 * (*inclinacion + inclinacion_giro) + 0.03 * inclinacion_acc;
           printf("La inclinacion es: %.3f\r\n",*inclinacion);
}

/**************************Fin Funciones SCU PRIVATE TIMER**********************/

void InicializaTimer(u16 TimerDeviceId,XScuTimer *TimerInstancePtr){
          XScuTimer_Config *ConfigPtr;
          ConfigPtr = XScuTimer_LookupConfig(TimerDeviceId);
          XScuTimer_CfgInitialize(TimerInstancePtr, ConfigPtr,ConfigPtr->BaseAddr);
}

void HandlerTimer(void *CallBackRef){
          /**********Variables para el cálculo Velocidad de las ruedas*********/
          static float time=0.01;
          static float rpm,rads;
          /*******Variables MPU6050*******/

          XScuTimer *TimerInstancePtr = (XScuTimer *) CallBackRef;
          XScuTimer_ClearInterruptStatus(TimerInstancePtr);
          num_rep++;
          /**********Variables del controlador**********/
          static float inclinacion=0;
          bool motor_on;
          float accion_control;
          /**********Cálculo de la velocidad**********/
          printf("Han pasado segundos %f desde el inicio:\r\n",num_rep/1000.0);
          vel_rueda1(time,&rpm,&rads);
          write_encoder_motor_en_1(2);
          printf("Este es el valor de la velocidad en la rueda derecha:\r\n rpm %.2f y rad/s %6.2f\r\n",rpm,rads);
          vel_rueda2(time,&rpm,&rads);
          write_encoder_motor_en_2(2);
          printf("Este es el valor de la velocidad en la rueda derecha:\r\n rpm %.2f y rad/s %6.2f\r\n",rpm,rads);
          mpu_ReadSensors(&mpu6050_i2c);
          FiltroComplementario(&inclinacion);
          PID(inclinacion,referencia,&accion_control,&motor_on);
          control_accion(accion_control,motor_on);
          mpu_printfAll();
          if (motor_on==true)
                    printf("\x1B[17A");
          else
                    printf("\x1B[13A");

}
```

```
void HabilitaSCUTimer(XScuGic *IntcInstancePtr, XScuTimer *TimerInstancePtr,u16 TimerIntrId){
                      XScuGic_Config *IntcConfig;
                      //Mira la configuracion del GIC
                      IntcConfig = XScuGic_LookupConfig(INTC_DEVICE_ID);
                      //Configura el SCU PT
                      XScuGic_CfgInitialize(IntcInstancePtr, IntcConfig,IntcConfig->CpuBaseAddress);
                      Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_IRQ_INT,(Xil_ExceptionHandler)XScuGic_InterruptHandler,
                                                   IntcInstancePtr);
                      //Conecta la interrupcion del SCU Private Timer con el HandlerTimer
                      XScuGic_Connect(IntcInstancePtr, TimerIntrId,(Xil_ExceptionHandler)HandlerTimer,
                                      (void *)TimerInstancePtr);
                      //Habilita el GIC
                      XScuGic_Enable(IntcInstancePtr, TimerIntrId);
                      //Habilita la interrupcion del SCU PrivateTimer
                      XScuTimer_EnableInterrupt(TimerInstancePtr);
                      //Habilita las interrupciones del micro
                      Xil_ExceptionEnable();

}

int main(void)
{

/*************Configuración MPu6050********/
            int status;
            status = initMPU6050IIC(&mpu6050_i2c,Config);
            if (status != XST_SUCCESS) {
            print("Ha fallado la inicialización del MPU6050\r\n");
            return XST_FAILURE;
            }

           //Inicializa el módulo SCU Private Timer
           InicializaTimer(TIMER_DEVICE_ID,&TimerInstance);
           //Habilita la interrupion del SCU Private Timer
           HabilitaSCUTimer(&IntcInstance,&TimerInstance,TIMER_IRPT_INTR);
           XScuTimer_EnableAutoReload(&TimerInstance);
           XScuTimer_LoadTimer(&TimerInstance, TIMER_LOAD_VALUE);
           XScuTimer_Start(&TimerInstance);
/*********Habilita los dos encoder************/
           write_encoder_motor_en_1(1);
           write_encoder_motor_en_2(1);
/************Habilita el driver del motor*************/
           driver_motor_en(1);
           while(1){

           }
           return 0;
}
```

ANEXO VI - IV Configuración del AXI IIC

Se ha configurado el I2C a 100KHz 7 bits de dirección.

Los valores 15 y 2 se corresponden con 300ns y 40ns segundos respectivamente de retraso hasta que se lee un pulso en las señales SCL y SDA. Esto es debido a que los pines de las Zynq son capaces de tal

velocidad que detectan todo tipo de ruidos que otros dispositivos no serían capaces de detectar.

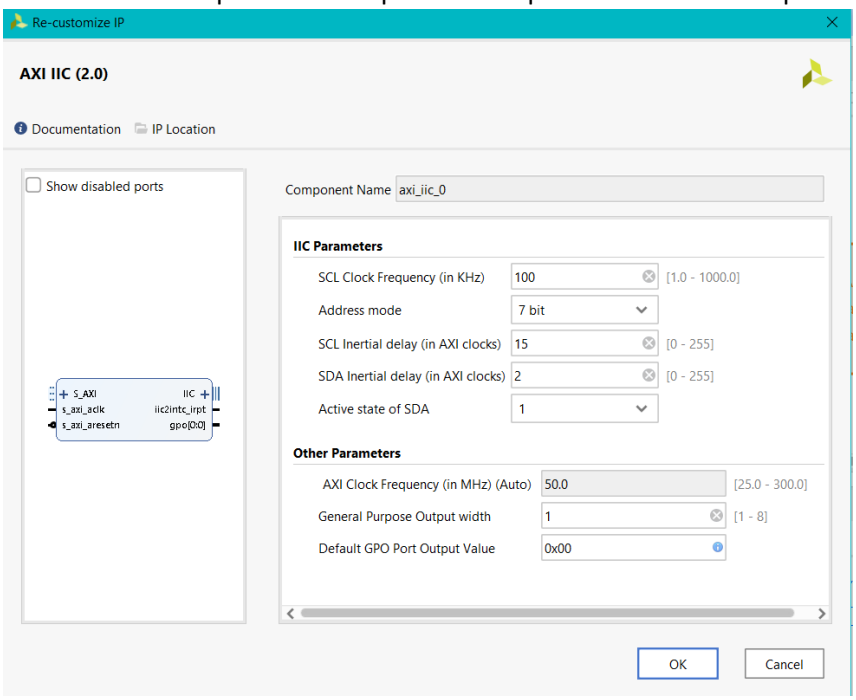


Figura 2-48 Cuadro de configuración del AXI IIC

ANEXO VI - V Configuración del CLK del PS

Esta configuración es importante porque determina a que velocidad irán los timers. Para concretar el timer privado del CPU funciona a la mitad del CPU. Es decir, como el CPU va a funcionar a 666.6666 MHz el SCU Private Timer funcionará a 333.3333MHz.

También se pueden ver los CLK que saca el bloque PS a la fábrica de la FPGA. En ella se puede comprobar que el FCLK_CLK0=50MHz y el FCLK_CLK1=10MHz.

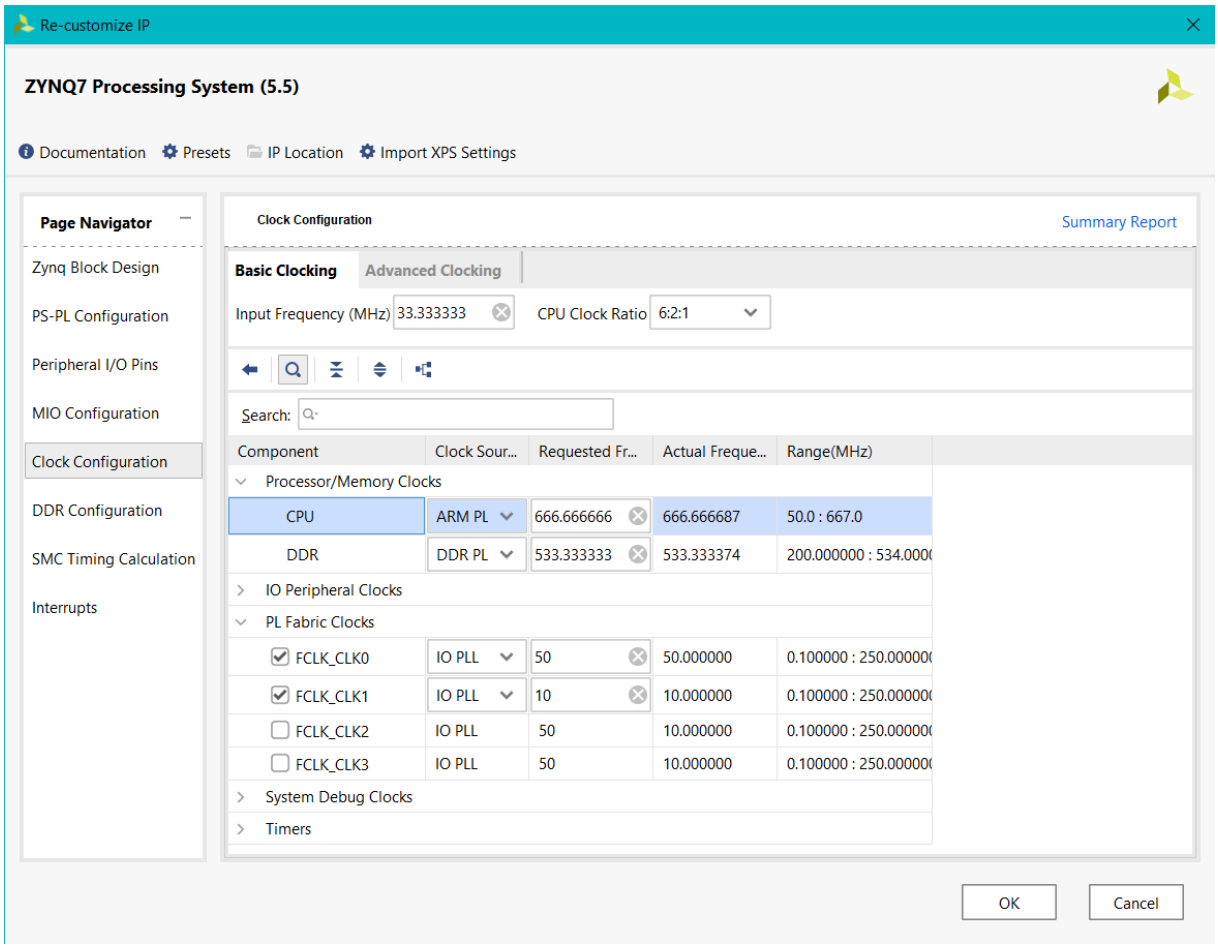
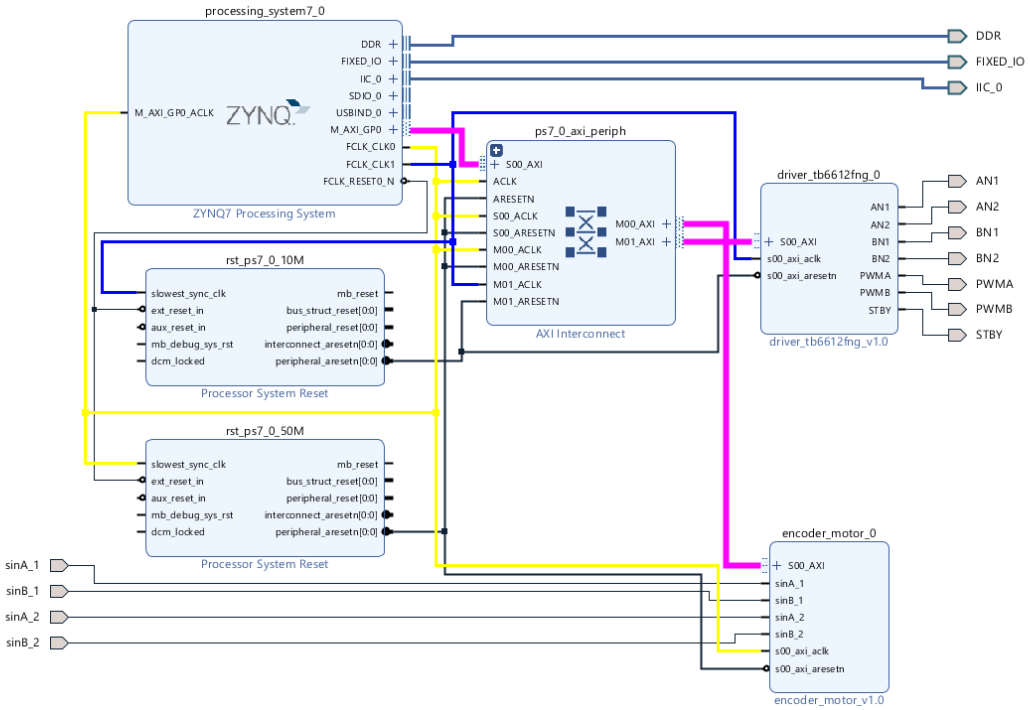
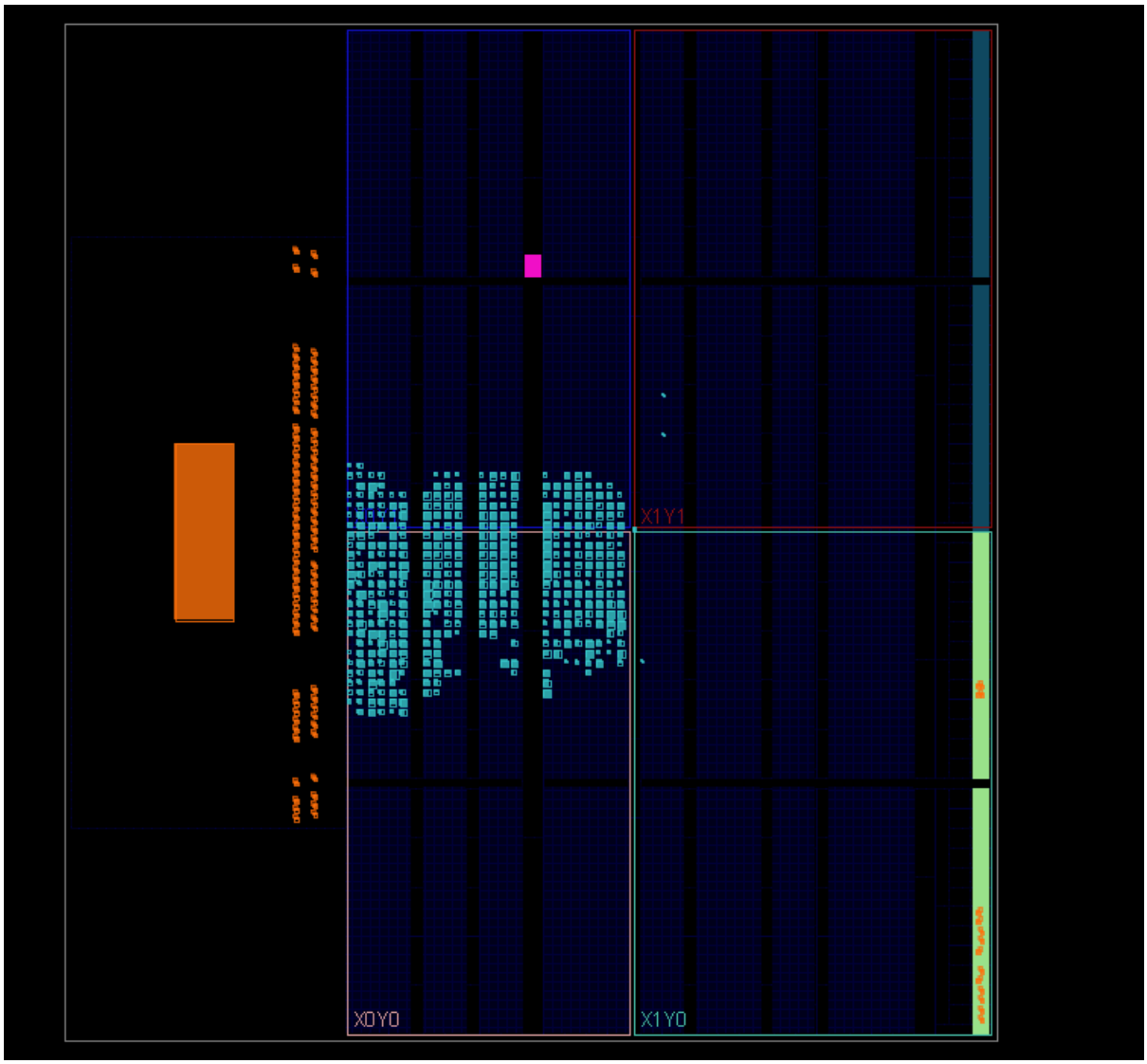


Figura 2-49 Bloque de configuración del PS, CLKs del sistema

ANEXO VI - VI Esquema de conexión hardware programable final



ANEXO VI - VII Implementación en el PL



ANEXO VI - VIII Constrains de la FPGA

```
# Bank 34
set_property PACKAGE_PIN R8  [get_ports {PWMA    }];  # "R8.ARDUINO_IO0"
set_property PACKAGE_PIN P8  [get_ports {AN1    }];  # "P8.ARDUINO_IO1"
set_property PACKAGE_PIN P9  [get_ports {AN2    }];  # "P9.ARDUINO_IO2"
set_property PACKAGE_PIN R7  [get_ports {PWMB    }];  # "R7.ARDUINO_IO3"
set_property PACKAGE_PIN N7  [get_ports {BN1    }];  # "N7.ARDUINO_IO4"
set_property PACKAGE_PIN R10 [get_ports {BN2    }];  # "R10.ARDUINO_IO5"
set_property PACKAGE_PIN P10 [get_ports {STBY    }];  # "P10.ARDUINO_IO6"
set_property PACKAGE_PIN N8  [get_ports {sinA_1    }];  # "N8.ARDUINO_IO7"
set_property PACKAGE_PIN M9  [get_ports {sinB_1    }];  # "M9.ARDUINO_IO8"
set_property PACKAGE_PIN N9  [get_ports {sinA_2    }];  # "N9.ARDUINO_IO9"
set_property PACKAGE_PIN M10 [get_ports {sinB_2   }];  # "M10.ARDUINO_IO10"
set_property PACKAGE_PIN P11 [get_ports {GPIO_0_tri_io[0]   }];  # "P11.ARDUINO_IO13"



# Bank 34
set_property PACKAGE_PIN M15 [get_ports {IIC_0_sda_io     }];  # "M15.PMOD1_D0_N"
set_property PACKAGE_PIN L15 [get_ports {IIC_0_scl_io     }];  # "L15.PMOD1_D0_P"
# Set the bank voltage for IO Bank 34 to 3.3V
set_property IOSTANDARD LVCMOS33 [get_ports -of_objects [get_iobanks 34]];
# Set the bank voltage for IO Bank 35 to 3.3V
set_property IOSTANDARD LVCMOS33 [get_ports -of_objects [get_iobanks 35]];
```

UNIVERSIDAD DE LA RIOJA

TRABAJO DE FIN DE GRADO

TITULACIÓN: GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

CURSO: 2019/2020 **CONVOCATORIA:** SEPTIEMBRE

TÍTULO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR HARDWARE PARA UN SISTEMA INESTABLE

AUTOR: ROOSBEL VINICIO GUAYA VEGA

DIRECTOR/ES: JAVIER ESTEBAN VICUÑA MARTÍNEZ Y JAVIER RICO AZAGRA

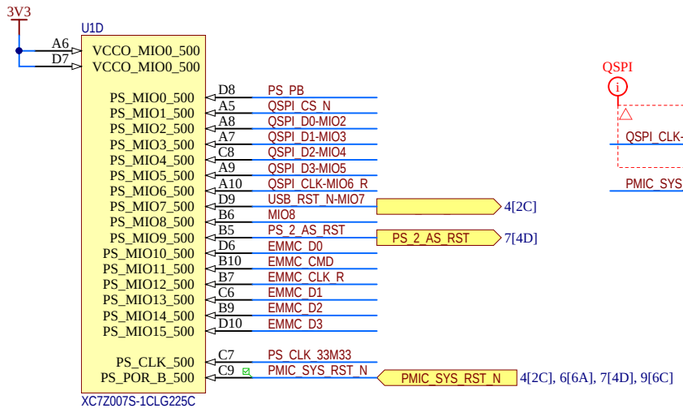
DEPARTAMENTO: INGENIERÍA ELÉCTRICA

ÍNDICE DE PLANOS

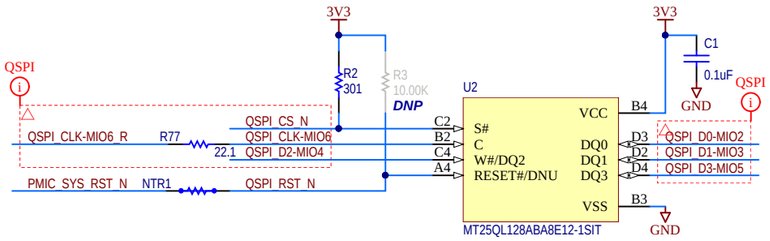
3 PLANOS

QSPI - eMMC - Boot Mode - PS Push Button

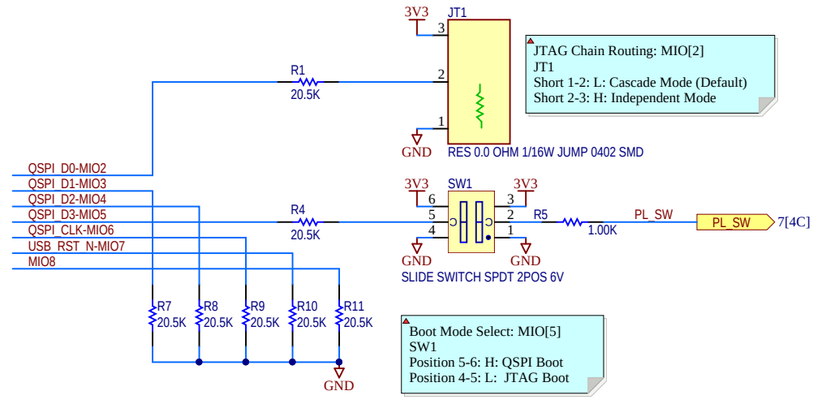
Zynq Bank500

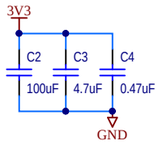


QSPI (QSPI0)

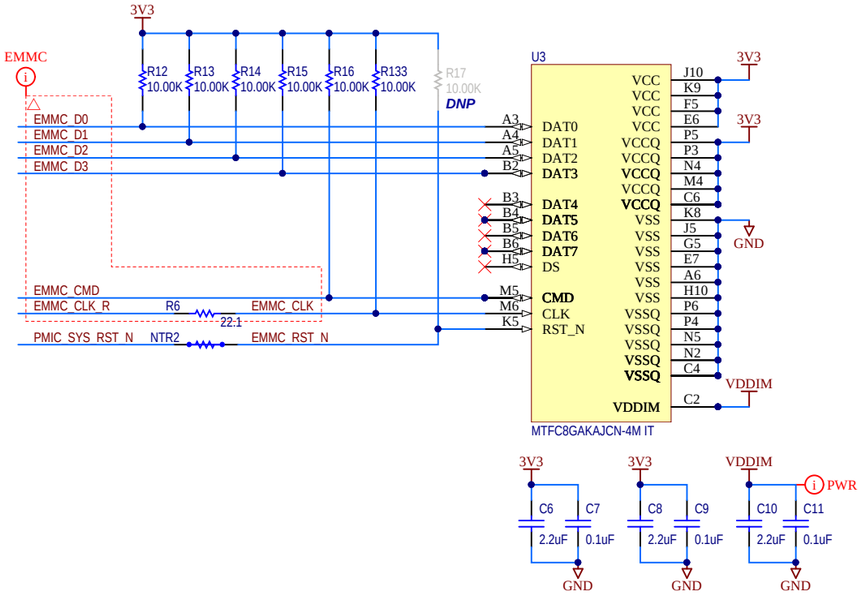


Boot Mode MIO Strapping Pins / PL User Switch

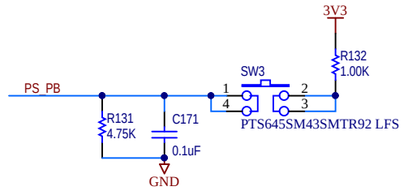


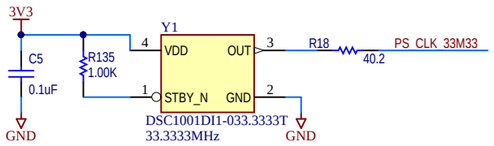


eMMC (SDIO1)



PS Push Button



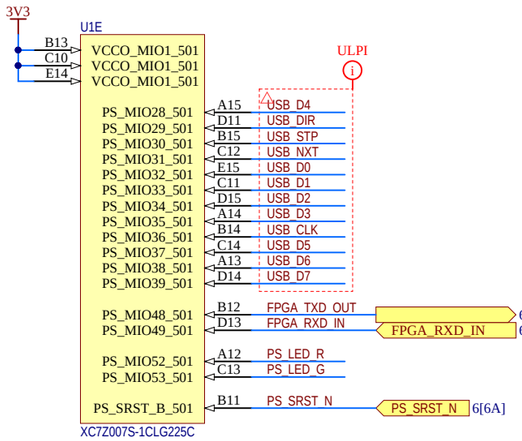


	FECHA	NOMBRE		ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL UNIVERSIDAD DE LA RIOJA
Dibujado	28/08/20	Roosbel Guaya		Grado en Ingeniería Electónica Industrial y Automática
Comprobado				
U.S.Norm.:	U.N.E.			

Escalas: SE	Diseño e implementación de un controlador hardware	Número: 1
PROYECCIÓN	Minized Schematic QSPI - eMMC - Boot Mode - PS Push Button	REFERENCIA: Sustituye a: Sustituido por:

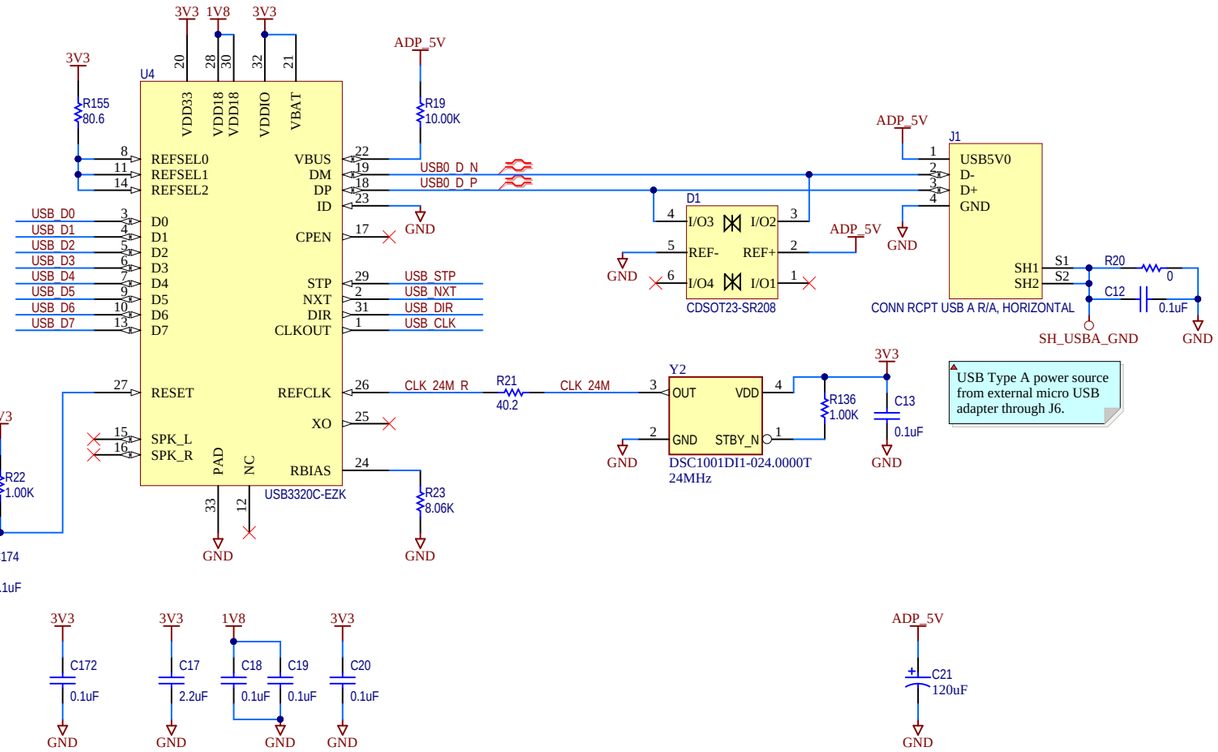
USB Host & PS LED

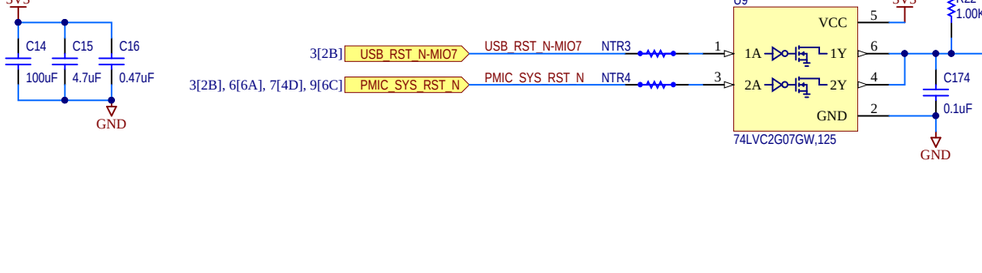
Zynq Bank501



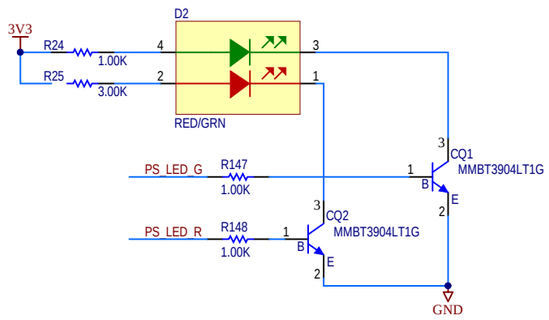
(UART1)

USB 2.0 Host (USB0)





PS User LEDs

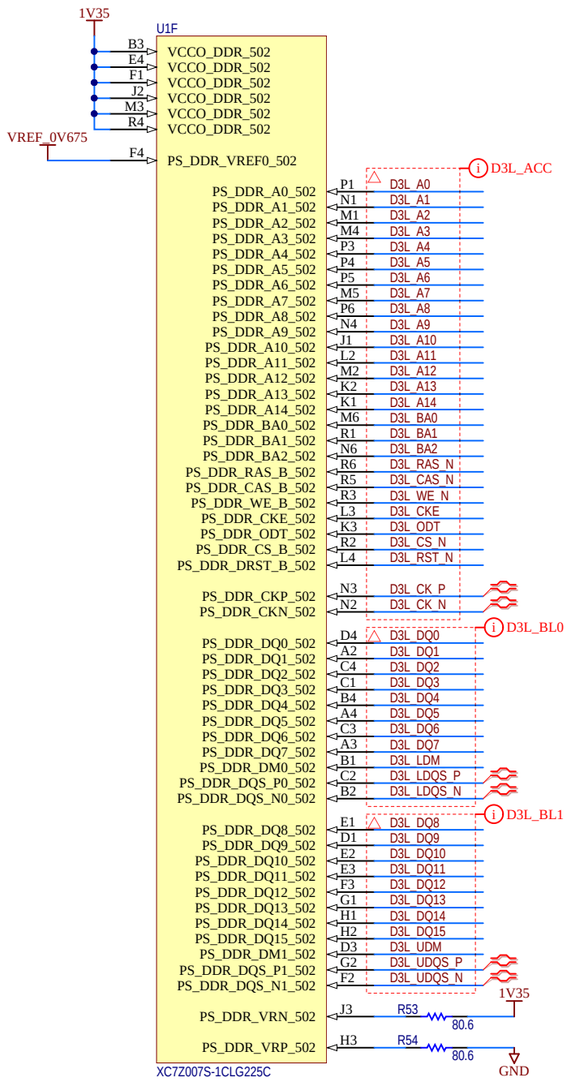


	FECHA	NOMBRE		ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL UNIVERSIDAD DE LA RIOJA
Dibujado	28/08/20	Roosbel Guaya		Grado en Ingeniería Electónica Industrial y Automática
Comprobado				
U.S.Norm.:	U.N.E.			

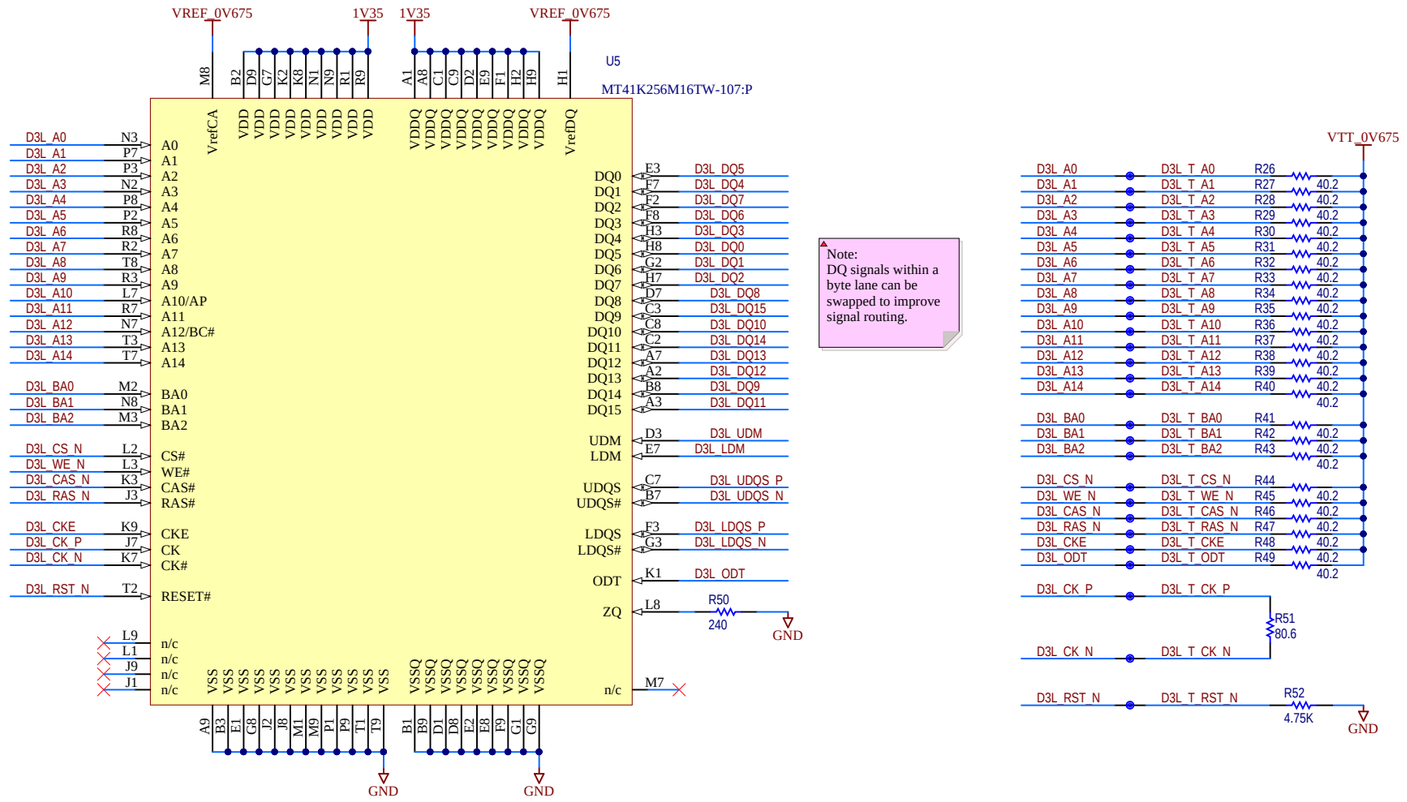
Escalas: SE	Diseño e implementación de un controlador hardware	Número: 2
PROYECCIÓN	Minized Schematic USB Host & PS LED	REFERENCIA: Sustituye a: Sustituido por:

DDR3L

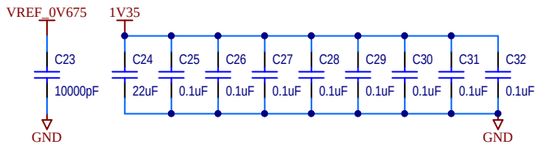
Zynq Bank502

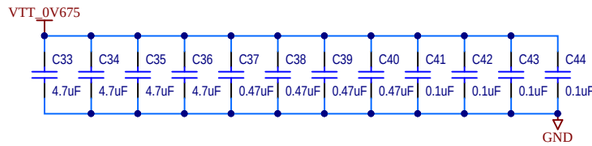


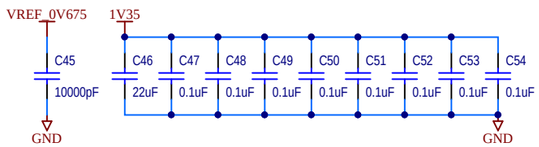
DDR3L

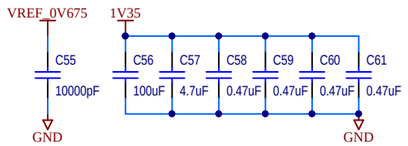


Note:
DQ signals within a byte lane can be swapped to improve signal routing.







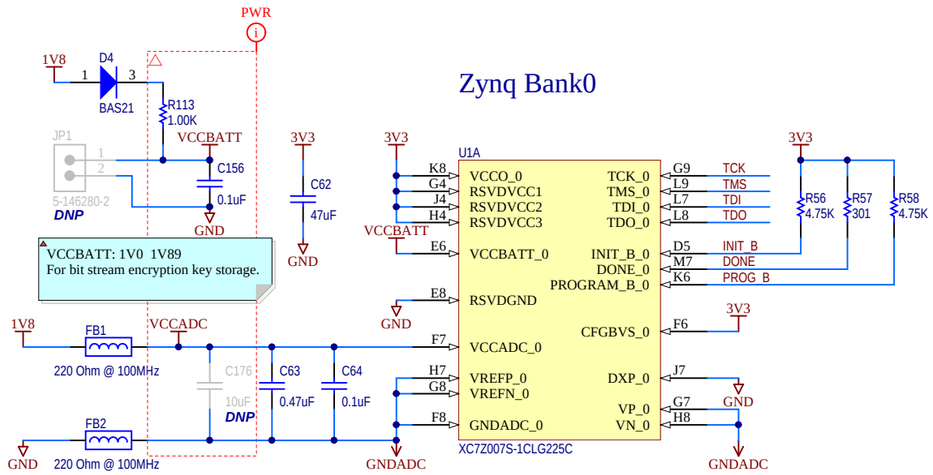


	FECHA	NOMBRE		ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL UNIVERSIDAD DE LA RIOJA
Dibujado	28/08/20	Roosbel Guaya		Grado en Ingeniería Electónica Industrial y Automática
Comprobado				
U.S.Norm.:	U.N.E.			

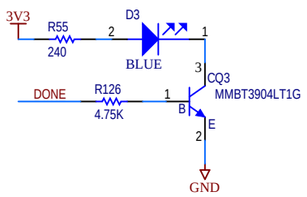
Escalas: SE	Diseño e implementación de un controlador hardware	Número: 3
PROYECCIÓN	Minized Schematic DDR3L	REFERENCIA: Sustituye a: Sustituido por:

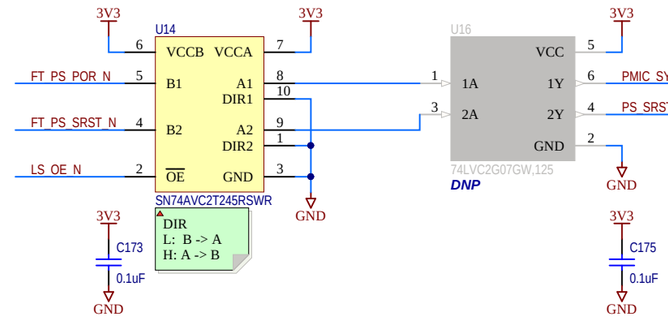
USB JTAG-UART

Zynq Bank0

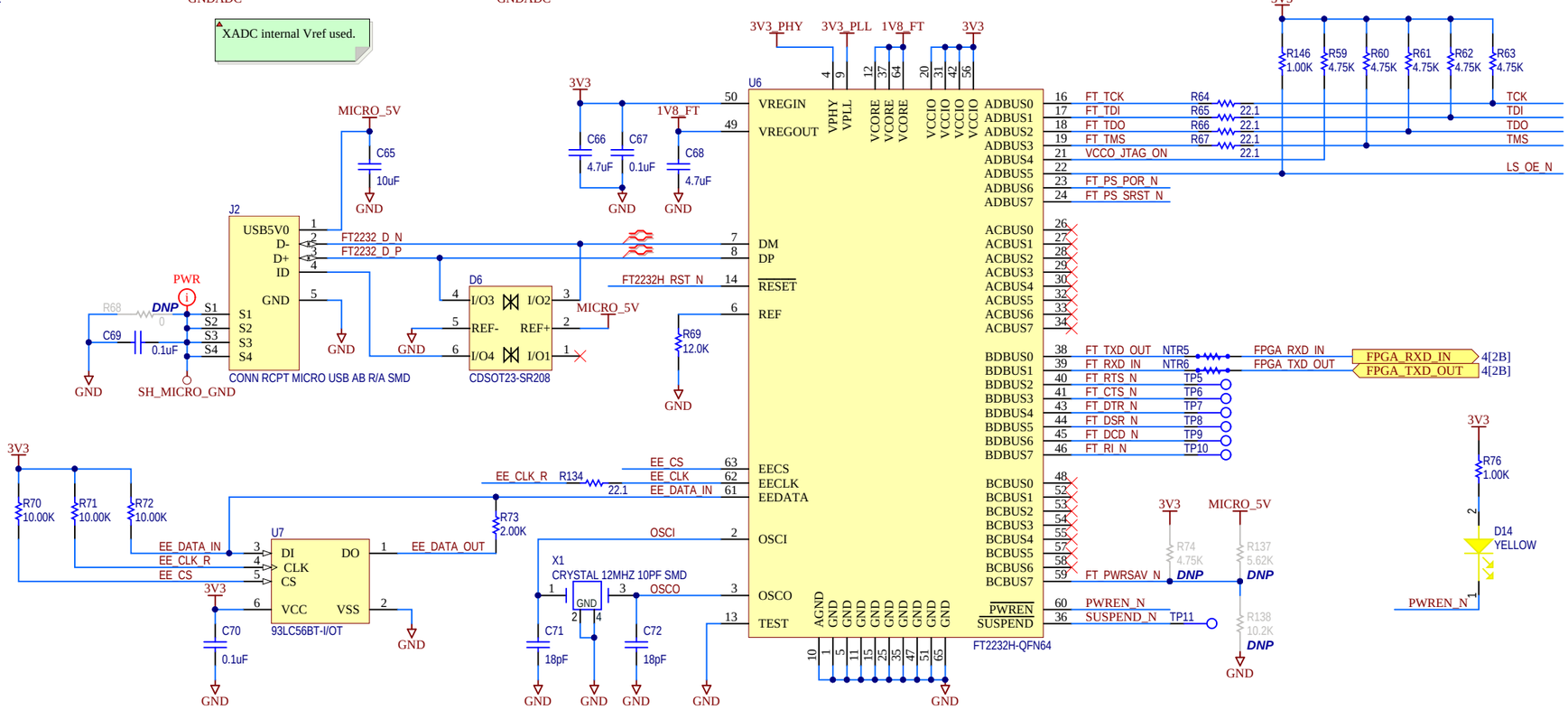


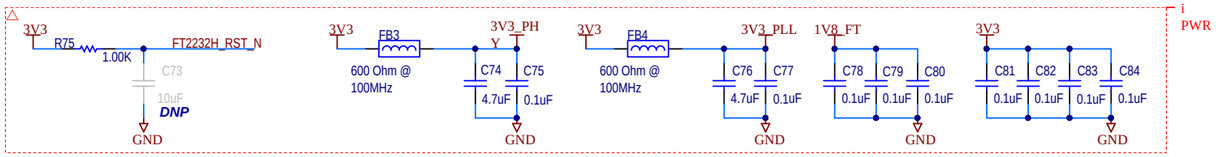
Done LED





USB JTAG & UART



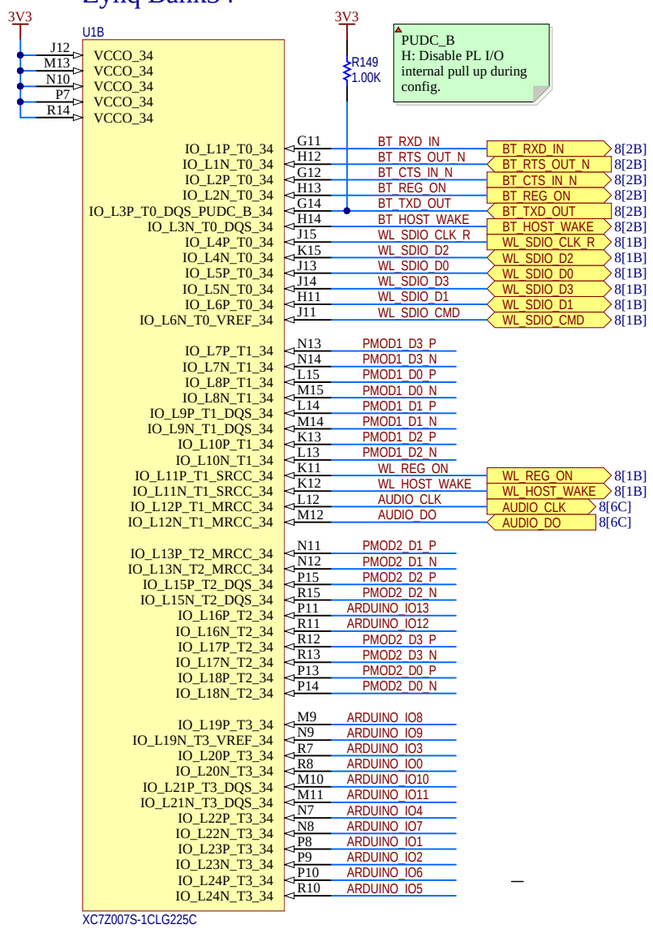


	FECHA	NOMBRE		ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL UNIVERSIDAD DE LA RIOJA
Dibujado	28/08/20	Roosbel Guaya		Grado en Ingeniería Electónica Industrial y Automática
Comprobado				
U.S.Norm.:	U.N.E.			

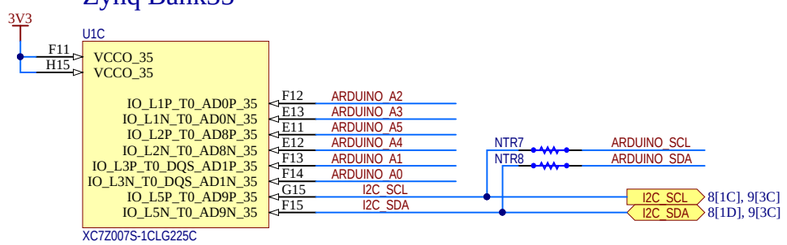
Escalas: SE	Diseño e implementación de un controlador hardware	Número: 4
PROYECCIÓN	Minized Schematic USB JTAG-UART	REFERENCIA: Sustituye a: Sustituido por:

Arduino - PMODs - PL LEDs

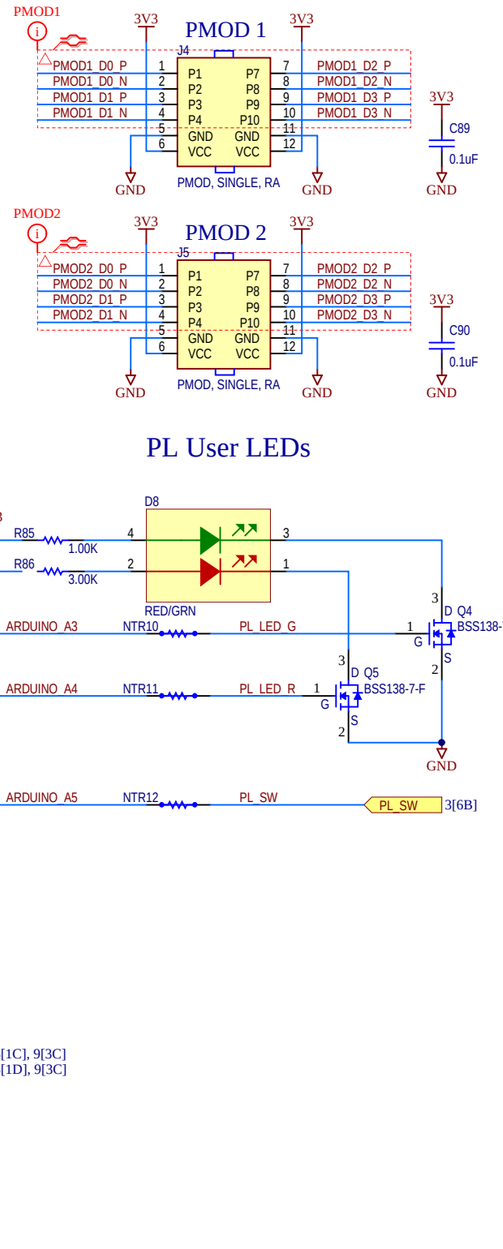
Zynq Bank34



Zynq Bank35

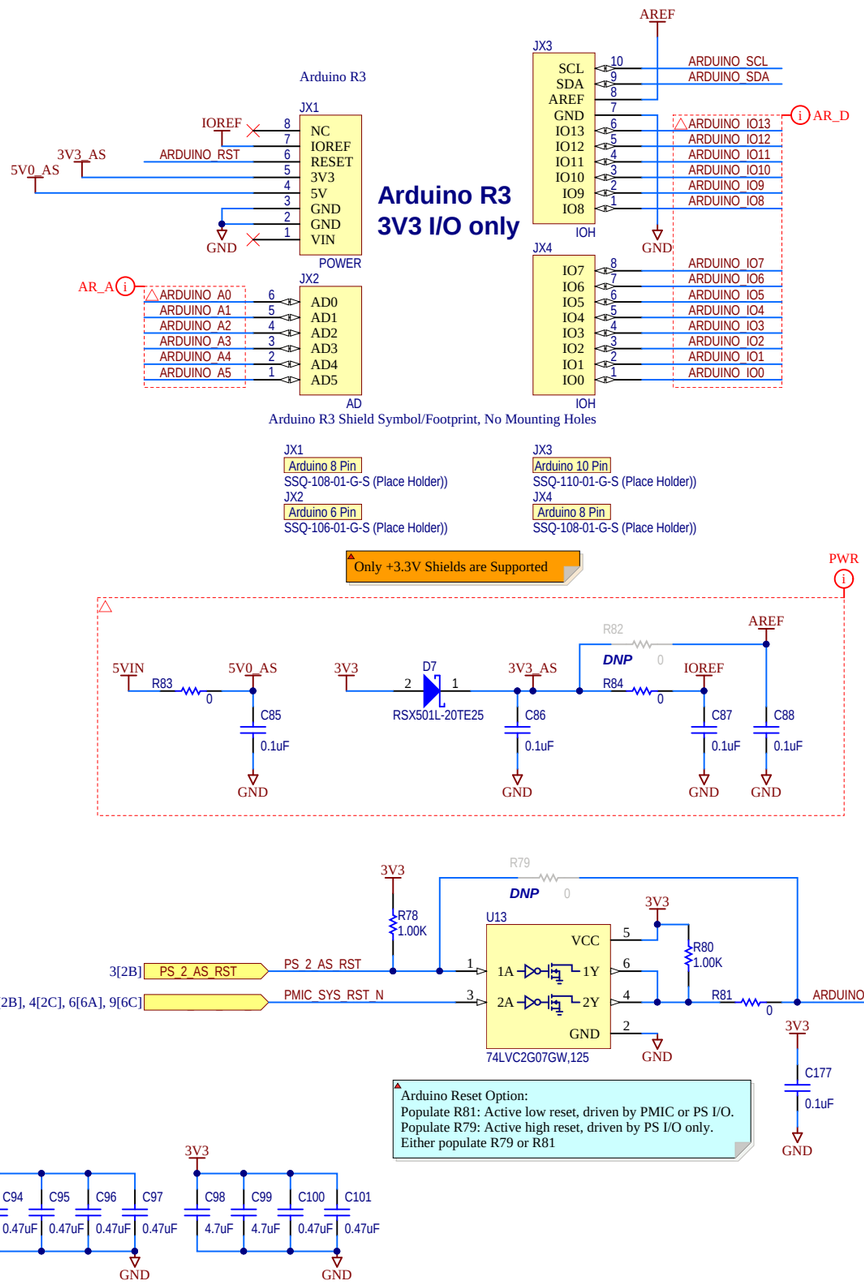


Dual PMODs



PL User LEDs

Arduino Shield

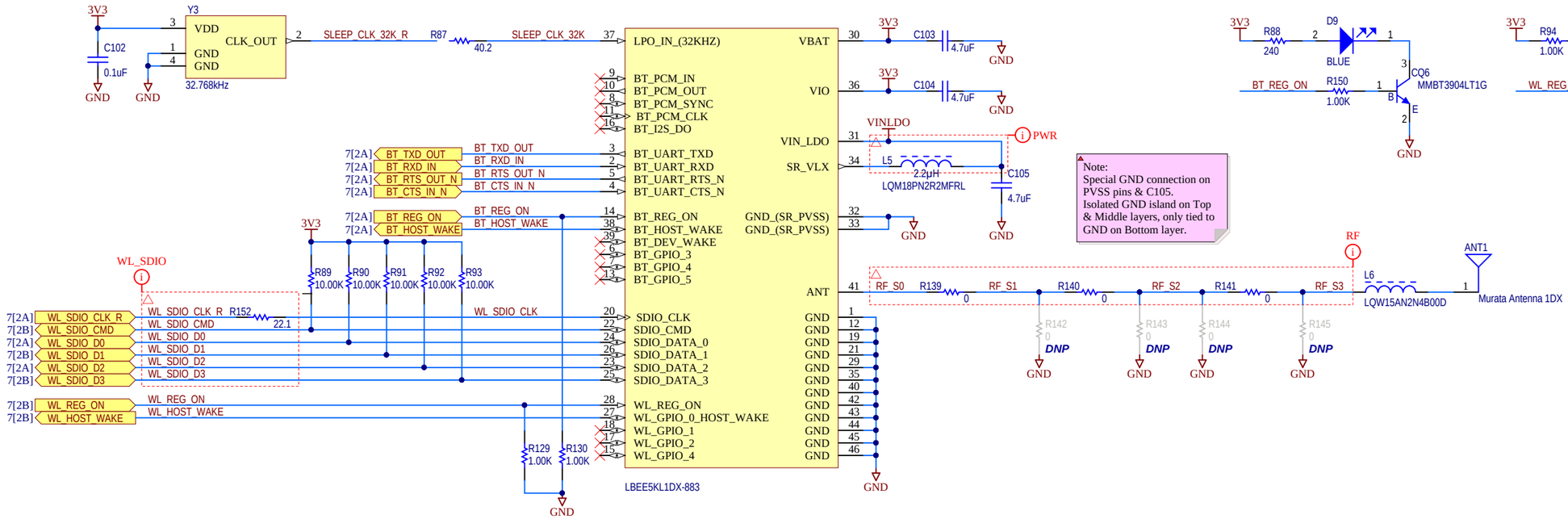


	FECHA	NOMBRE		ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL UNIVERSIDAD DE LA RIOJA
Dibujado	28/08/20	Roosbel Guaya		Grado en Ingeniería Electónica Industrial y Automática
Comprobado				
U.S.Norm.:	U.N.E.			

Escalas:	Diseño e implementación de un controlador hardware	Número: 5
SE	Minized Schematic	REFERENCIA:
PROYECCIÓN	Arduino - PMODs - PL LEDs	Sustituye a:
		Sustituido por:

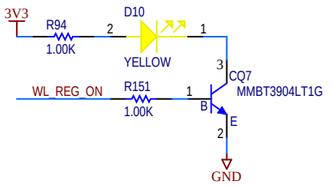
Wireless - Sensors

WiFi & Bluetooth Module

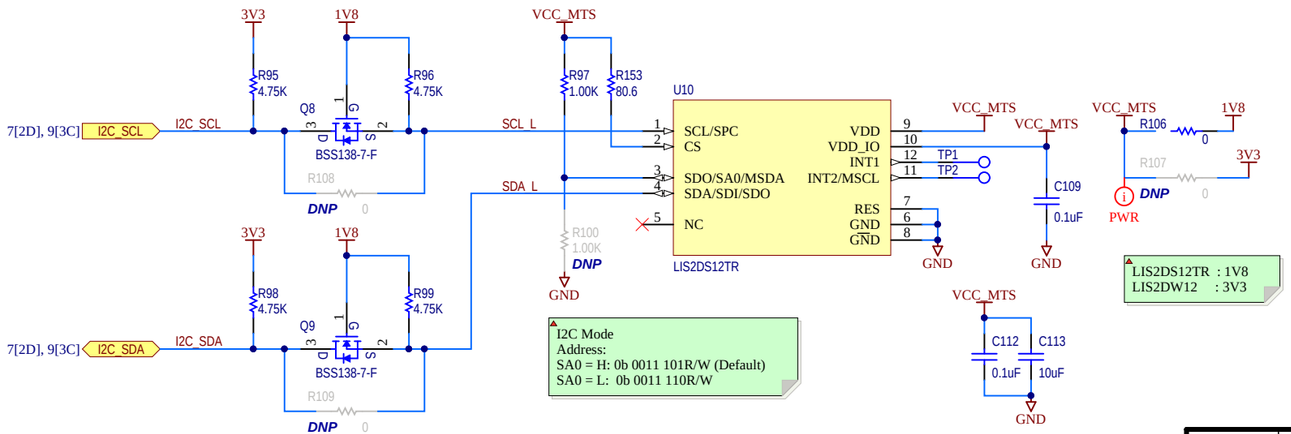


Bluetooth Power On

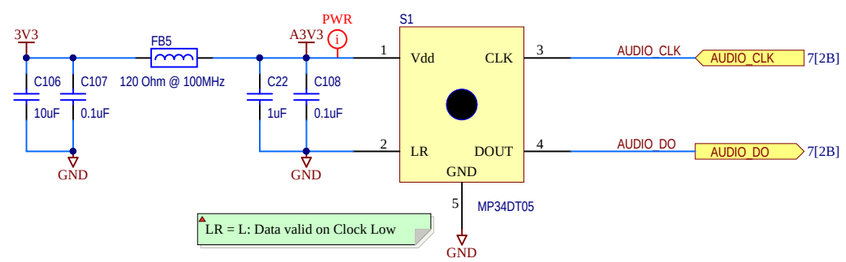
WiFi Power On



Motion & Temperature Sensor



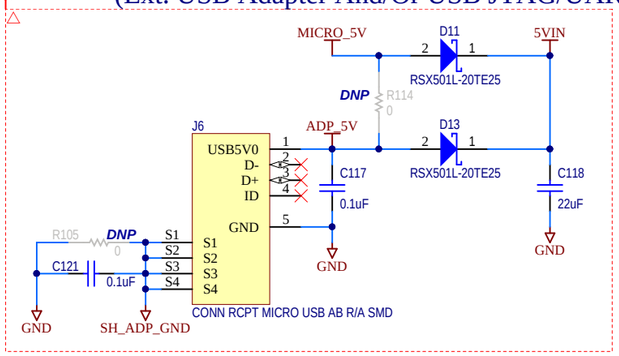
Audio Sensor



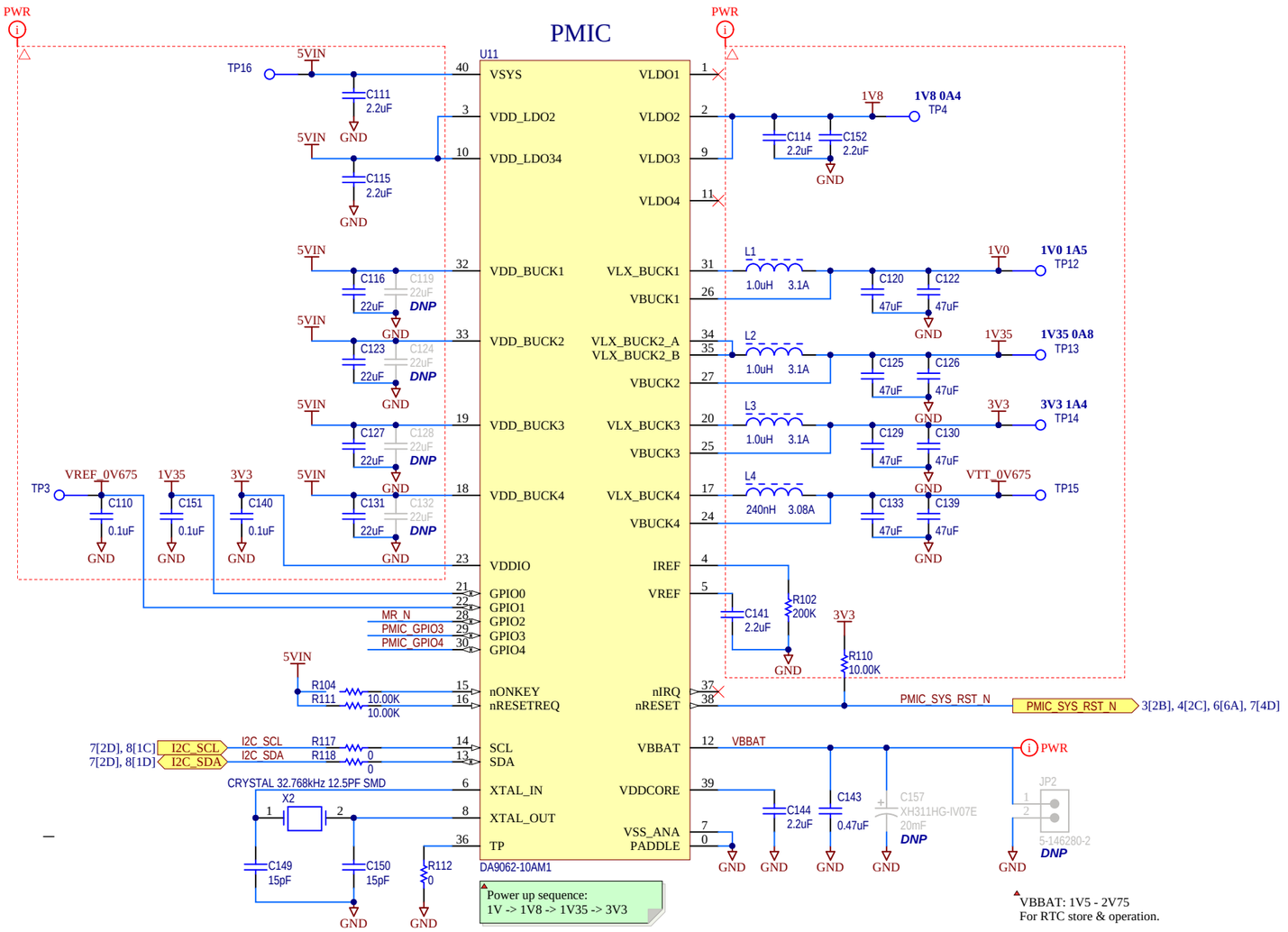
	FECHA	NOMBRE		ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL UNIVERSIDAD DE LA RIOJA Grado en Ingeniería Electónica Industrial y Automática	
Dibujado	28/08/20	Roosbel Guaya			
Comprobado					
U.S.Norm.:	U.N.E.				
Escalas: SE	Diseño e implementación de un controlador hardware				Número: 6
PROYECCIÓN	Minized Schematic Wireless comm- Sensors				REFERENCIA: Sustituye a: Sustituido por:

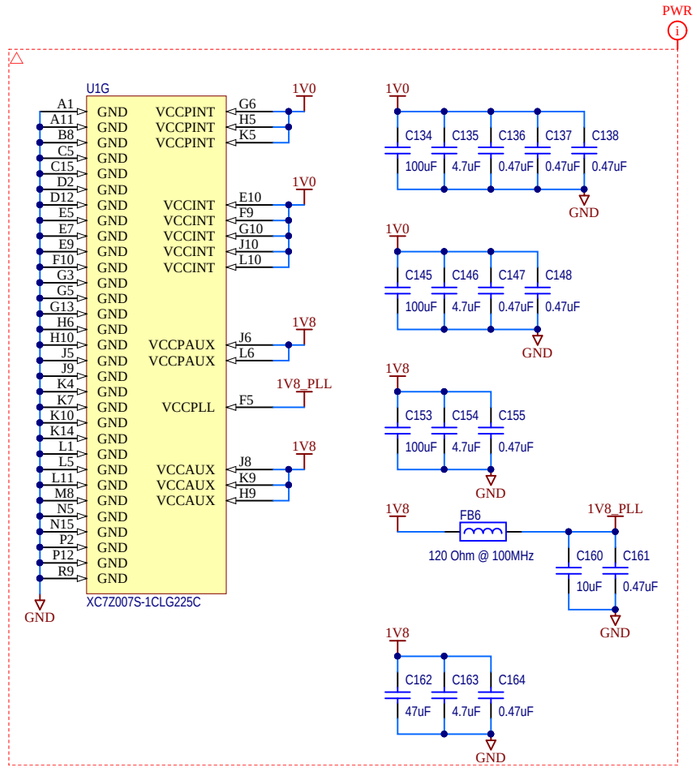
Power Supplies

Micro USB port power
(Ext. USB Adapter And/Or USB JTAG/UART)

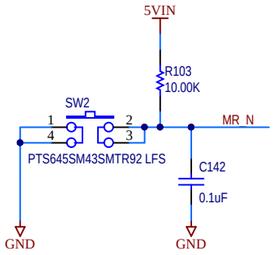


PMIC

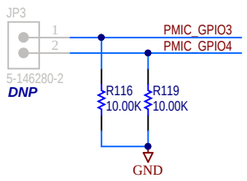




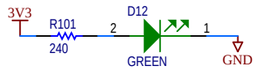
RESET PUSHBUTTON



PMIC GPIO

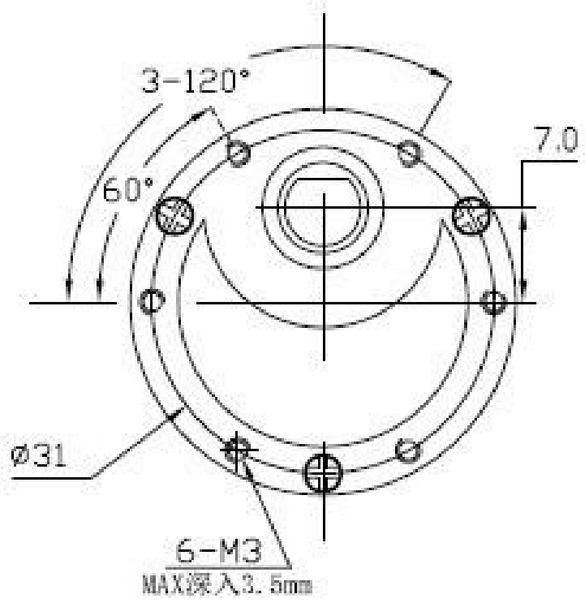


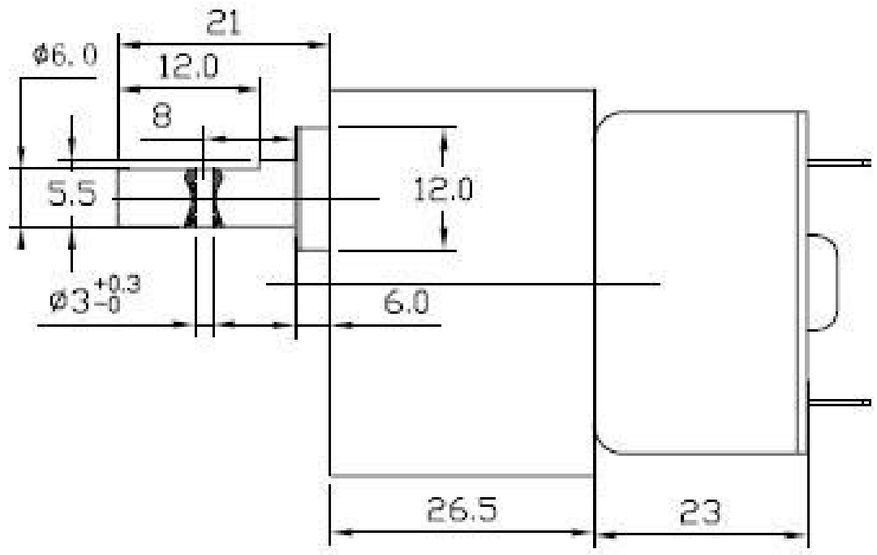
POWER READY LED

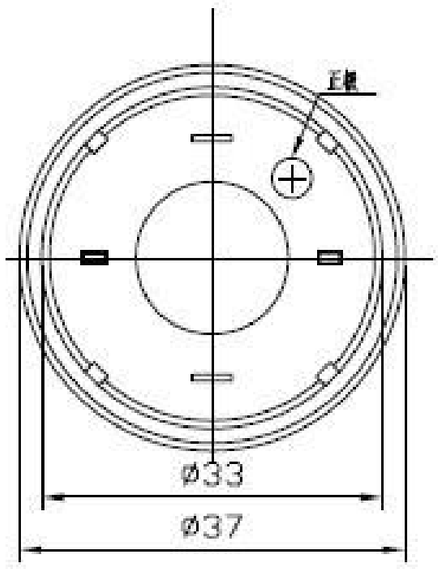


	FECHA	NOMBRE		ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL UNIVERSIDAD DE LA RIOJA
Dibujado	28/08/20	Roosbel Guaya		Grado en Ingeniería Electónica Industrial y Automática
Comprobado				
U.S.Norm.:	U.N.E.			

Escalas: SE	Diseño e implementación de un controlador hardware	Número: 7
PROYECCIÓN	Minized Schematic Power Supplies	REFERENCIA: Sustituye a: Sustituido por:

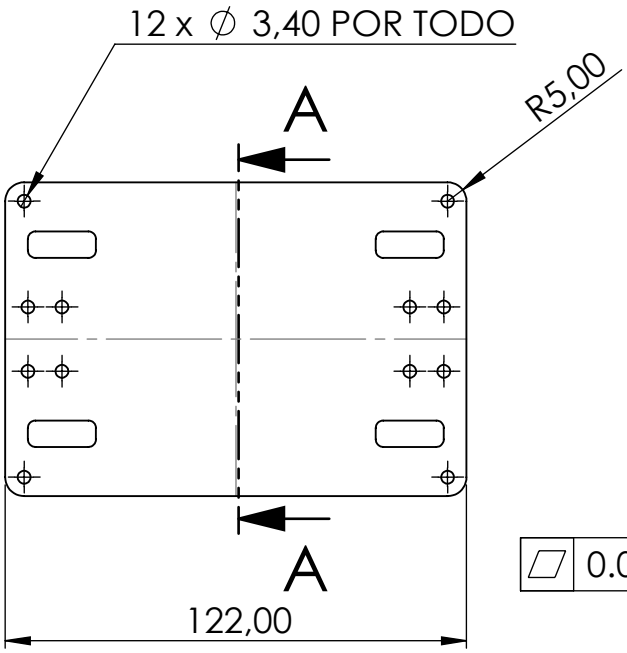


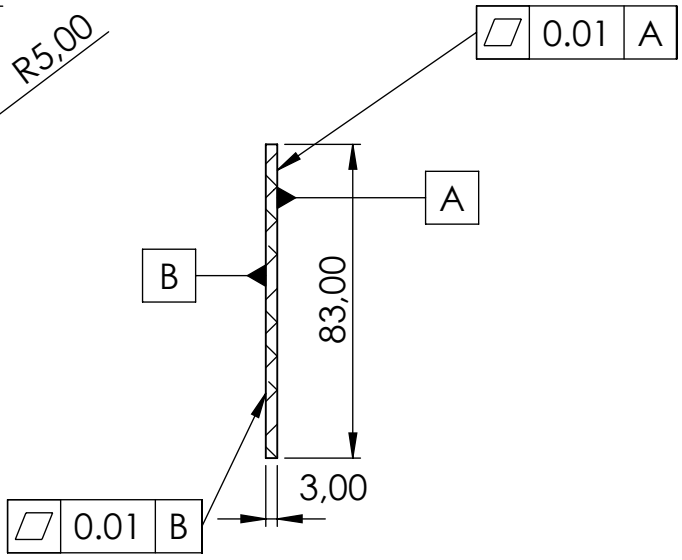




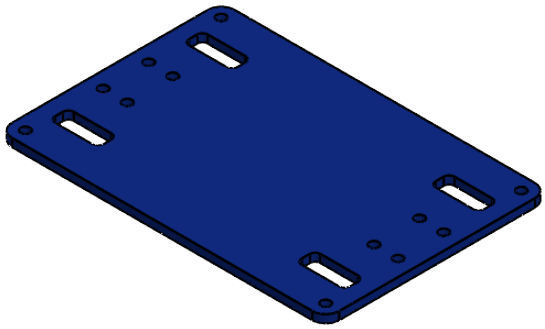
	FECHA	NOMBRE		ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL UNIVERSIDAD DE LA RIOJA Grado en Ingeniería Electónica Industrial y Automática
Dibujado	28/08/20	Roosbel Guaya		
Comprobado				
U.S.Norm.:	U.N.E.			

Escalas: SE	Diseño e implementación de un controlador hardware	Número: 8
PROYECCIÓN	Minized Schematic Motor Eléctrico CC	REFERENCIA: Sustituye a: Sustituido por:



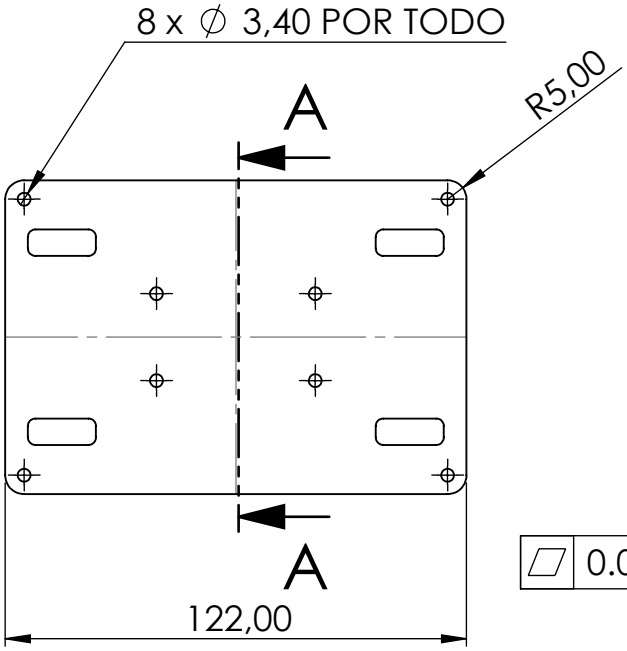


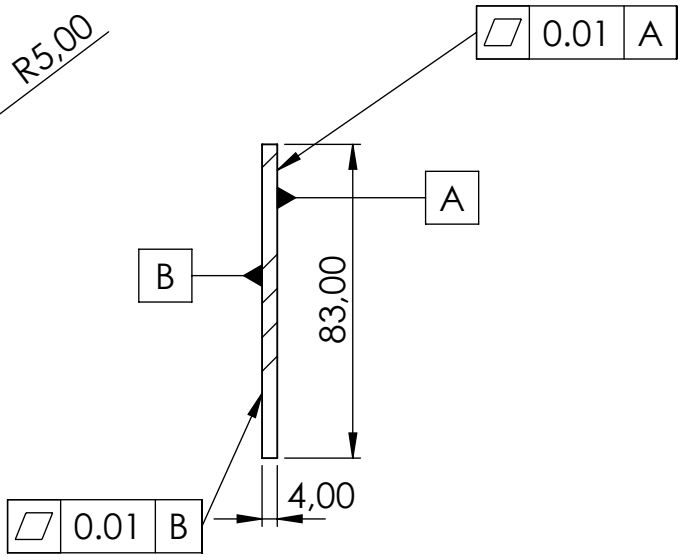
SECCIÓN A-A



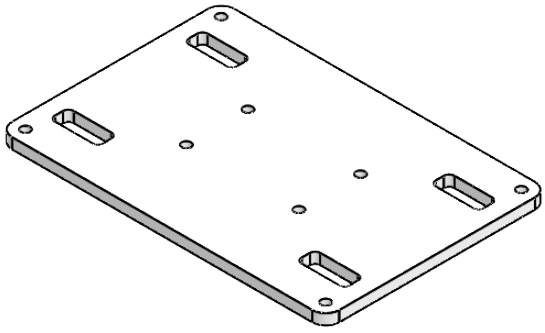
	FECHA	NOMBRE		ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL UNIVERSIDAD DE LA RIOJA GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA
Dibujado	02/09/2020	Miguel Mart. de Laguna		
Comprobado		Roosbel Guaya		
Id.s.normas	UNE			

ESCALAS 1:2	Diseño e implementación de un controlador hardware	Número 9
Proyección	Chapa Inferior	Referencia Sustituye a Sustituido por





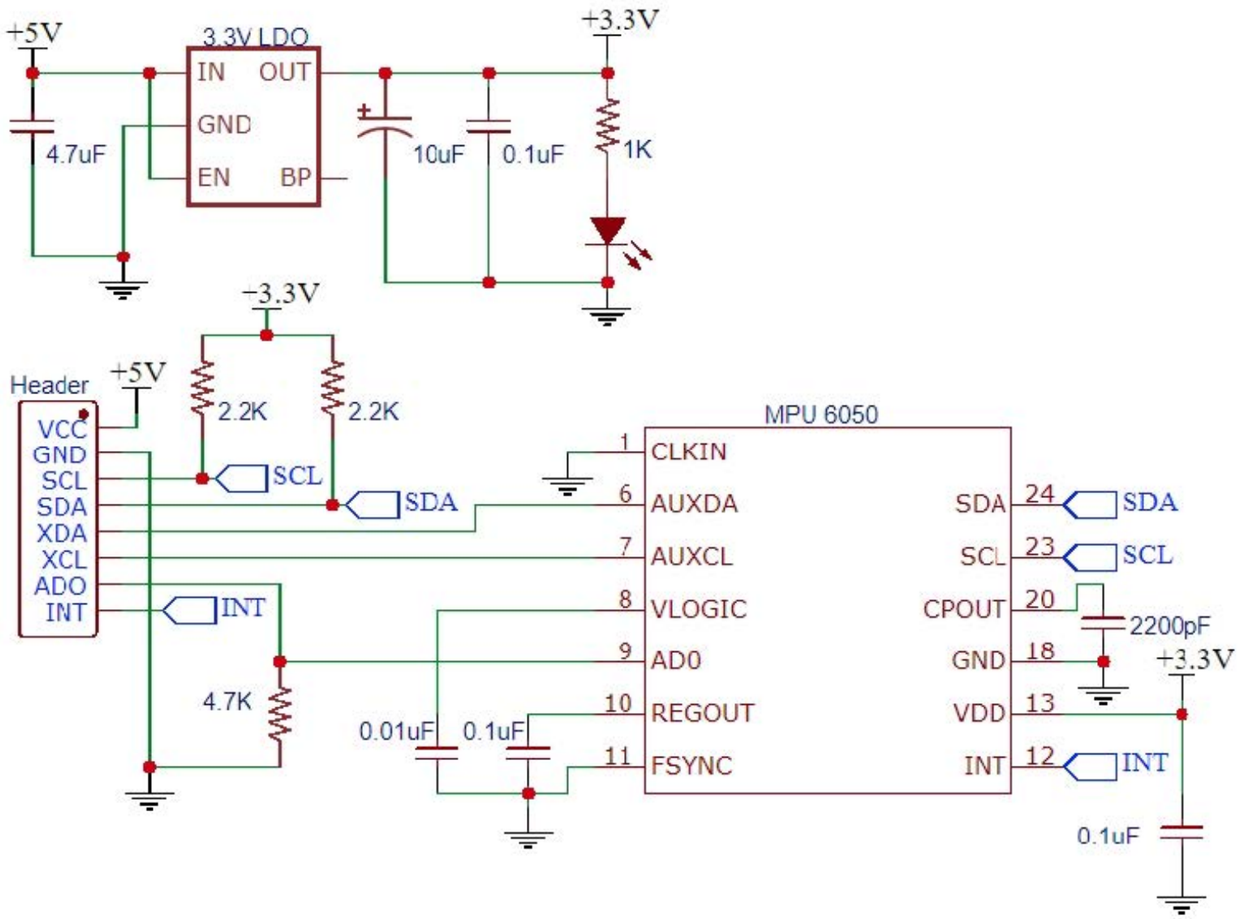
SECCIÓN A-A



	FECHA	NOMBRE		ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL UNIVERSIDAD DE LA RIOJA GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA
Dibujado	02/09/2020	Miguel Mart. de Laguna		
Comprobado		Roosbel Guaya		
Id.s.normas	UNE			

ESCALAS	Diseño e implementación de un controlador hardware	Número
1:2		10
Proyección	Chapa Media y Superior	Referencia
		Sustituye a
		Sustituido por

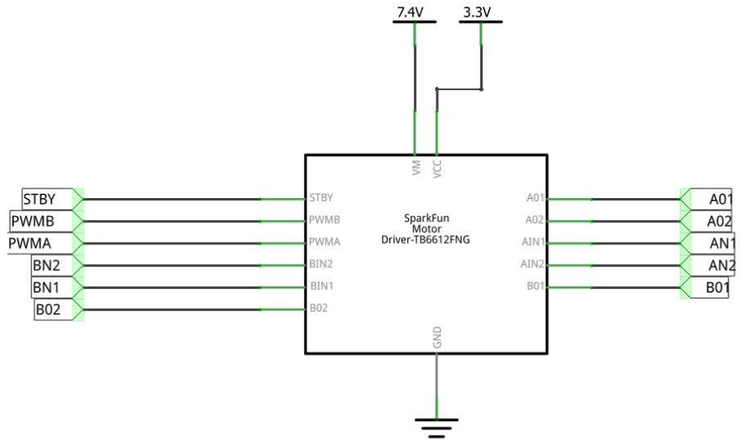
Esquema del MPU6050

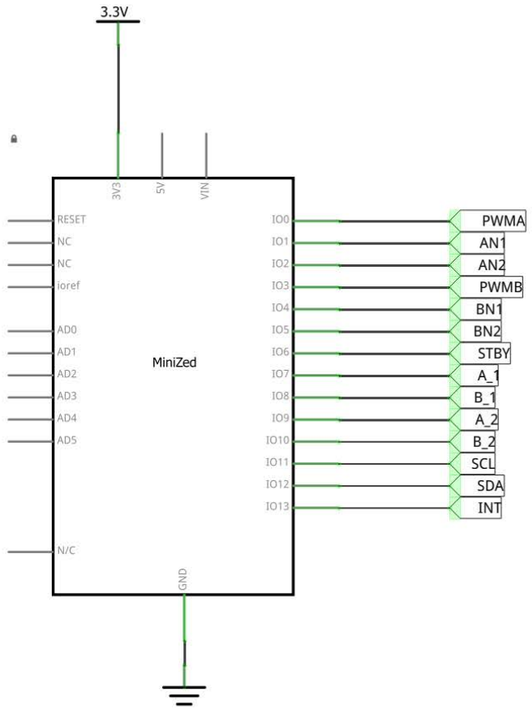


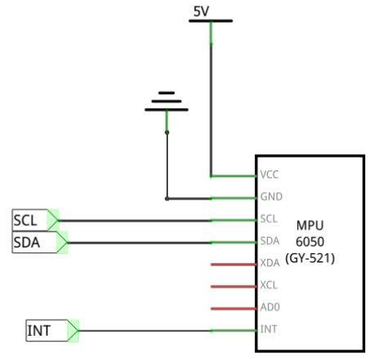
	FECHA	NOMBRE		ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL UNIVERSIDAD DE LA RIOJA Grado en Ingeniería Electónica Industrial y Automática
Dibujado	28/08/20	Roosbel Guaya		
Comprobado				
U.S.Norm.:	U.N.E.			

Escalas: SE	Diseño e implementación de un controlador hardware	Número: 11
PROYECCIÓN	Esquema del MPU6050	REFERENCIA: Sustituye a: Sustituido por:

Esquema de conexión Minized-MPU6050-TB6612FNG



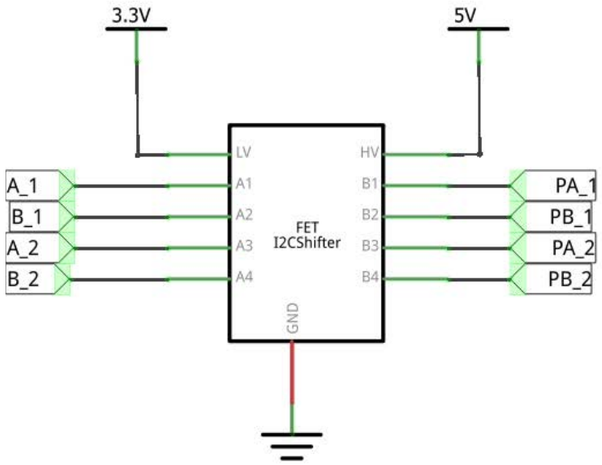


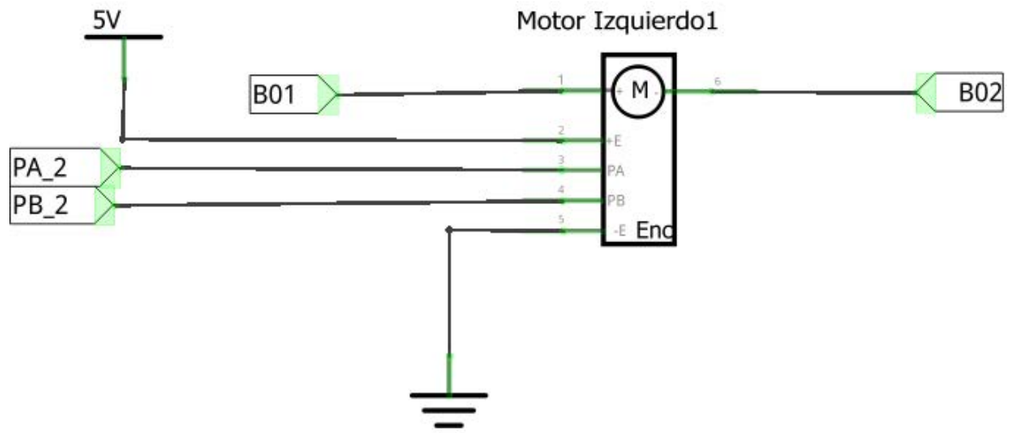


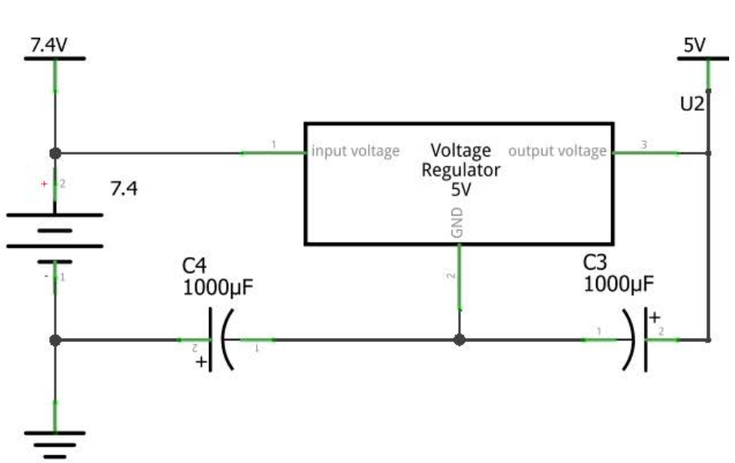
	FECHA	NOMBRE		ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL UNIVERSIDAD DE LA RIOJA Grado en Ingeniería Electónica Industrial y Automática
Dibujado	28/08/20	Roosbel Guaya		
Comprobado				
U.S.Norm.:	U.N.E.			

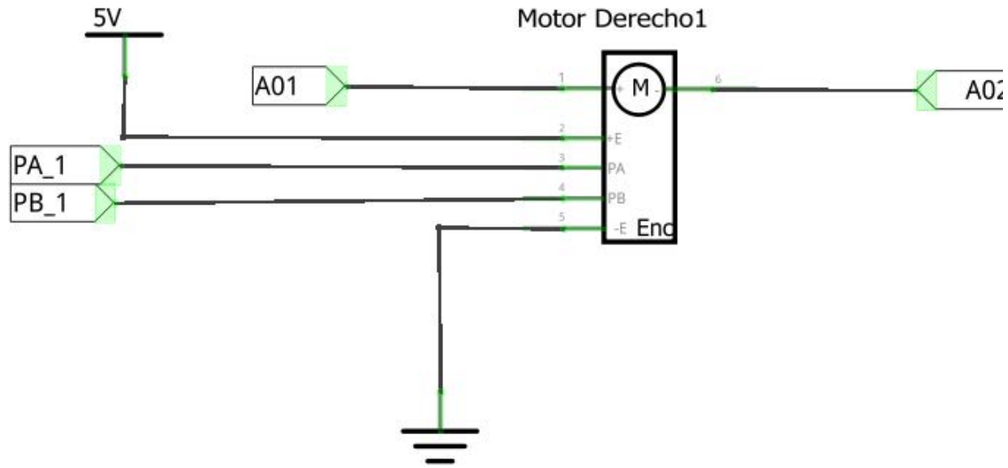
Escalas: SE	Diseño e implementación de un controlador hardware	Número: 12
PROYECCIÓN	Esquema de conexión Minized-MPU6050-TB6612FNG	REFERENCIA: Sustituye a: Sustituido por:

Esquema de conexión Alimentación-LevelShifter-Motores









	FECHA	NOMBRE		ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INDUSTRIAL UNIVERSIDAD DE LA RIOJA Grado en Ingeniería Electónica Industrial y Automática
Dibujado	28/08/20	Roosbel Guaya		
Comprobado				
U.S.Norm.:	U.N.E.			

Escalas: SE	Diseño e implementación de un controlador hardware	Número: 13
PROYECCIÓN	Esquema de conexión Alimentación-LevelShifter-Motores	REFERENCIA: Sustituye a: Sustituido por:

UNIVERSIDAD DE LA RIOJA

TRABAJO DE FIN DE GRADO

TITULACIÓN: GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

CURSO: 2019/2020 **CONVOCATORIA:** SEPTIEMBRE

TÍTULO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR HARDWARE PARA UN SISTEMA INESTABLE

AUTOR: ROOSBEL VINICIO GUAYA VEGA

DIRECTOR/ES: JAVIER ESTEBAN VICUÑA MARTÍNEZ Y JAVIER RICO AZAGRA

DEPARTAMENTO: INGENIERÍA ELÉCTRICA

4 PLIEGO DE CONDICIONES

4.1.1 LISTADO DE MATERIALES

Elemento	Material
Chapa inferior	ALUM 5754-H22
Chapas superiores	Policarbonato Compacto Transparente

Tabla 4-1 Listado de materiales

4.1.2 ESPECIFICACIONES DE LOS MATERIALES

A continuación, se exponen las propiedades mecánicas de los materiales que componen.

- ALUM 5754

Composición:

Componente	% Present
Aluminium	94.2 - 97.4
Chromium	0.3 max
Copper	0.1% max
Iron	0.4% max
Magnesium	2.6 - 3.6%
Manganese	0.5% max
Silicon	0.4% max
Titanium	0.15% max
Zinc	0.2% max
Residuals	0.15% max

Tabla 4-2 Composición ALUM 5754 Fuente: [8]

Propiedades mecánicas:

Propiedad mecánica	Valor
Tensión de fluencia	130 MPa
Tensión última	220 - 270 MPa
Dureza Brinell	63 HB
Alargamiento unitario	7 %

Tabla 4-3 Propiedades mecánicas ALUM 5754 Fuente: [8]

- Policarbonato Compacto Transparente

Propiedad mecánica	Valor
Módulo elástico(E)	2960 MPa
Coeficiente de poisson	0,37
Densidad	1420 kg/m^3
Tensión última a la tracción	57.3 MPa
Tensión última a la compresión	92.9 MPa

Tabla 4-4 Propiedades mecánicas del PET Fuente: SOLIDWORKS

4.2 CARACTERÍSTICAS DE LOS EQUIPOS

En este apartado se recogen los equipos que se han usado para fabricar los componentes del robot, en especial las chapas que forman parte del esqueleto estructural del mismo. Se hace una breve descripción del equipo, pero se excluye la maquinaría que ha sido necesaria para definir los elementos adquiridos a terceros como por ejemplo: el motor eléctrico, las ruedas, los casquillos, la sujeción en L o las tuercas alargadas, pues cada uno de ellos tienen su fabricación que depende explícitamente del fabricante.

- Fresadora

Esta máquina permite crear superficies planas y rectas mediante el movimiento rotativo de la herramienta. En este proyecto se usará para aplanar las chapas y obtener la forma deseada.

- Taladro de mesa

Permite taladrar distintas superficies planas para definir las distintas sujeciones atornilladas.

En este proyecto es necesario crear los agujeros para sujetar los motores a la sujeción en L con la chapa inferior. También es necesario para hacer los agujeros que sujetarán las tuercas alargadas con las chapas.

4.3 CALIDADES MÍNIMAS DE LOS ELEMENTOS

La calidad mínima está asociada a las tolerancias dimensionales y geométricas, así como al estado superficial de cada una de las piezas. Estas tolerancias permiten que el robot encaje en su construcción y no tenga problemas en cuanto a la alineación de sus ruedas. En los siguientes apartados se explica las tolerancias requeridas.

4.3.1 TOLERANCIAS DIMENSIONALES Y GEOMÉTRICAS

La calidad mínima de los elementos no solamente está asociado al estado superficial de cada una de las piezas, sino que también se deben cumplir con las tolerancias dimensionales y geométricas establecidas para la correcta colocación de los elementos que componen el Robot, por ello en este apartado se establecen un conjunto de tolerancias de índole dimensional o geométrica para lograr el citado fin.

Tolerancias geométricas y dimensionales en las chapas

Se propone para las chapas una tolerancia geométrica de planitud en las secciones planas tanto superior como inferior de (118x83mm) de 0,01mm, con ello se pretende facilitar la colocación de los elementos electrónicos como puedan ser la batería o la protoboard, además se establece una coaxialidad mínima de 0,01 en las uniones entre las tuercas alargadas y las mencionadas chapas, para permitir el correcto montaje, además de reducir las tensiones que se ocasionarían en el caso de eludir dicha tolerancia en el montaje.

4.4 EJECUCIÓN DEL PROYECTO

En este apartado se explica el procedimiento para la ejecución del proyecto, donde se explica cómo se ha de desarrollar los componentes y especificaciones de fabricación de los elementos que componen el robot.

Cada componente será realizado por personal cualificado e instruido para ello y, cumpliendo con las normas de seguridad.

Todas las medidas y tolerancias, tanto dimensionales como geométricas, tienen que coincidir con las exigencias de los planos.

Chapa inferior de aluminio:

La chapa inferior está formada por una lámina de aluminio que se conforma de la siguiente manera:

1. La lámina de aluminio de medidas 130x90x5mm se le pasa la fresadora hasta conseguir el contorno de la chapa de 122x83x3mm.
2. Seguidamente se realizan distintas operaciones de taladrado de los agujeros métrica 3.

Todo ello atendiendo a las tolerancias geométricas anteriormente citadas.

Chapas superiores de PET:

La chapa superior está formada por una lámina de Policarbonato Compacto Transparente que se conforma de la siguiente manera:

1. La lámina de PET de medidas 130x90x4mm se le pasa la fresadora hasta conseguir el contorno de la chapa de 122x83x4mm.
2. Seguidamente se realizan distintas operaciones de taladrado de los agujeros métrica 3.

Todo ello atendiendo a las tolerancias geométricas anteriormente citadas.

4.5 MONTAJE DEL ROBOT

El montaje del robot empieza atornillando las rocas alargadas M3x45mm a la chapa inferior de aluminio. Se atornillarán las cuatro roscas alargadas M3x45mm para después poner sobre ellas la chapa superior de PET. Sobre esta chapa hay dos tornillos en el centro que serán usados para atornillar el sujetador de las baterías. Sobre la chapa superior de PET se atornillarán las 4 tuercas alargadas de M3x25mm y sobre ellas se atornillará la última chapa de PET.

Al cada motor se atornillarán los cuatro tornillos a la sujeción en L en la cara de su eje. En el eje del motor se atornillará el casquillo y después se unirá ese casquillo a una rueda por la parte interna. Por la parte externa de la rueda en el eje se hace pasar un tornillo para que sujete bien la rueda y no se salga.

Se procederá atornillando la sujeción en L a la parte inferior de la chapa de aluminio con cuatro tornillos a cada lado de la chapa. Los ejes de los motores unidos a las ruedas tienen que señalar al exterior.

A la protoboard que ha sido ya conectada con todos sus componentes, se le quitará la protección inferior para que se pegue a la parte superior de la chapa de aluminio.

UNIVERSIDAD DE LA RIOJA

TRABAJO DE FIN DE GRADO

TITULACIÓN: GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

CURSO: 2019/2020 **CONVOCATORIA:** SEPTIEMBRE

TÍTULO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR HARDWARE PARA UN SISTEMA INESTABLE

AUTOR: ROOSBEL VINICIO GUAYA VEGA

DIRECTOR/ES: JAVIER ESTEBAN VICUÑA MARTÍNEZ Y JAVIER RICO AZAGRA

DEPARTAMENTO: INGENIERÍA ELÉCTRICA

5 MEDICIONES

5.1 PLACAS

	Placa de aluminio		
N.º	Descripción	Medición	Ud.
1.1	kg, Corte de chapa de aluminio AL-5754 para su posterior fresado (130x90x5mm).	0,146	kg
1.2	Ud. fresado del contorno de la chapa (122x83x3mm) de aluminio según características de diseño, incluye el proceso la realización de agujeros para los tornillos necesarios mediante un taladro de mesa.	1	Ud.

Tabla 5-1 Mediciones placa de aluminio

	Placa de PET (Policarbonato compacto transparente)		
N.º	Descripción	Medición	Ud.
2.1	kg, Corte de chapa de Policarbonato compacto transparente o (PET) de para su posterior fresado (130x90x4mm)	0,0406	kg
2.2	Ud., fresado del contorno de la chapa de Policarbonato compacto transparente o (PET) según características de diseño, incluye el proceso la realización de agujeros para los tornillos necesarios mediante un taladro de mesa	1	Ud.

Tabla 5-2 Mediciones placa de PET

5.2 MOTORES

	Motor eléctrico CC		
N.º	Descripción	Medición	Ud.
3.1	Ud. Motor eléctrico de corriente continua (CC) de potencia 4,8W y 12V con reductora de velocidad i=1/30	2	Ud.

Tabla 5-3 Mediciones motor eléctrico CC

5.3 RUEDAS

	Ruedas de coche de juguetes de Ø65mm		
N.º	Descripción	Medición	Ud.
4.1	Ud. Ruedas fabricadas mediante inyección de plástico	2	Ud.

Tabla 5-4 Mediciones ruedas

5.4 ELEMENTOS ELCTRÓNICOS

	Batería		
N.º	Descripción	Medición	Ud.
5.1	Ud. Batería Li-on Samsung INR18650-35E 3450mAh	2	Ud.
5.2	Ud. Carcasa de baterías holder	1	Ud.

	Protoboard		
N.º	Descripción	Medición	Ud.
6.1	Ud. Protoboard	1	Ud.

	MPU		
Nº	Descripción	Medición	Ud.
7.1	Ud. MPU 6050	1	Ud.

	TB6612FNG		
N.º	Descripción	Medición	Ud.
8.1	Ud. TB6612FNG	1	Ud.

	Level Shifter AN10441		
Nº	Descripción	Medición	Ud.
9.1	Ud. Level Shifter (5V - 3,3V) AN10441	1	Ud.

	Regulador lineal L7805CV		
Nº	Descripción	Medición	Ud.
10.1	Ud. Regulador lineal de 5V L7805CV	1	Ud.

	MiniZed		
N.º	Descripción	Medición	Ud
11.1	Ud. MiniZed	1	Ud.

Tabla 5-5 Mediciones elementos electrónicos

5.5 ELEMENTOS DE MEDICIÓN

	Logic Analyzer		
N.º	Descripción	Medición	Ud.
12.1	Ud. Logic Analyzer	1	Ud.

	Oscilloscope Hantek 6022-BE		
N.º	Descripción	Medición	Ud.
13.1	Ud. Oscilloscope Hantek 6022-BE	1	Ud.

Tabla 5-6 Mediciones elementos de medición

5.6 TORNILLERÍA

	Tornillería		
Nº	Descripción	Medición	Ud.
T01	Ud. Tornillo de cabeza plana M3x5mm	18	Ud.
T02	Ud. Tuerca alargada de M3x45mm	4	Ud.
T03	Ud. Tuerca alargada de M3x25mm	4	Ud.
T04	Ud. Casquillo pasador de bronce	2	Ud.

Tabla 5-7 Mediciones tornillería

UNIVERSIDAD DE LA RIOJA

TRABAJO DE FIN DE GRADO

TITULACIÓN: GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

CURSO: 2019/2020 **CONVOCATORIA:** SEPTIEMBRE

TÍTULO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR HARDWARE PARA UN SISTEMA INESTABLE

AUTOR: ROOSBEL VINICIO GUAYA VEGA

DIRECTOR/ES: JAVIER ESTEBAN VICUÑA MARTÍNEZ Y JAVIER RICO AZAGRA

DEPARTAMENTO: INGENIERÍA ELÉCTRICA

6 PRESUPUESTO

6.1 CUADRO DE MAQUINARIA

Núm.	Código	Denominación de la maquinaria	Precio	Cantidad	Ud.	Total (€)
1	MA01	Fresadora	10€/h	1	h	10,00 €
2	MA02	Taladro de mesa	5€/h	0,33	h	1,65 €

Tabla 6-1 Presupuesto cuadro de maquinaria

6.2 CUADRO DE MATERIAL

Núm.	Código	Denominación del material	Precio	Cantidad	Ud.	Total (€)
1	M01	Aluminio AL-5057	5€/kg	0,146	kg	0,73 €
2	M02	PET Policarbonato compacto transparente	10€/kg	0,0812	kg	0,81 €
3	M03	Ud, Motor eléctrico de corriente continua (CC) de potencia 4,8W y 12V con reductora de velocidad i=1/30	7,71€/Ud.	2	Ud.	15,42 €
4	M04	Ud, Ruedas fabricadas mediante inyección de plástico	4,18€/Ud.	2	Ud.	8,36 €
5	M05	Batería	14,9€/Ud.	1	Ud.	14,90 €
6	M06	Protoboard	2,60€/Ud.	1	Ud.	2,60 €
7	M07	MPU 6050	2,76€/Ud.	1	Ud.	2,76 €
8	M08	TB6612FNG	4,19€/Ud.	1	Ud.	4,19 €
9	M09	Level Shifter AN10441	3,79€/Ud.	1	Ud.	3,79 €
10	M10	Regulador lineal L7805CV	3,36€/Ud.	1	Ud.	3,36 €
11	M11	MiniZed	107€/Ud.		Ud.	107,00 €
12	M12	Analizador Lógico	15,28€/Ud.		Ud.	15,28 €
13	M13	Osciloscopio	78,12€/Ud.		Ud.	78,12 €
14	M14	Tornillería	3€/Ud.		Ud.	3,00 €

Tabla 6-2 Cuadro de material

6.3 CUADRO DE MANO DE OBRA

Núm.	Código	Mano de obra	Precio	Cantidad	Ud.	Total (€)
1	P01	Operario de taller de fresado y ajuste manual	15 €	1,33	h	19,95 €
2	P02	Operario para montaje del robot	20 €	0,5	h	10 €
3	P03	Programación Hardware	100€	150	h	15000 €
4	P04	Programación Software	50€	100	h	5000 €
5	P05	Modelado y simulación en Matlab	50€	50	h	2500 €
6	P06	Diseño mecánico en Solid Works	50€	8	h	400€

Tabla 6-3 Cuadro de mano de obra

6.4 Licencias de Software

Software Informático	Precio (€)
Matlab	2000 €
Simulink	3000 €
Matlab Control ToolBox	1075 €
Vivado design suite-HL System Edition	3500 €
Sigasi	865 €
Microsoft Office 360	240 €
SolidWorks	10950 €
Total:	21.630,00 €

6.5 PRECIO UNITARIO MATERIAL

	Importe
1.Placas	33,14 €
2. Motores	15,42 €
3.Ruedas	8,36 €
4.Batería	14,90 €
5.Protoboard	2,60 €
6.MPU	2,76 €
7.TB6612FNG	4,19 €
8.Level Shifter	3,79 €
9.Regulador lineal	3,36 €
10.Minzed	107,00 €
11.Tornillería	3,00 €
12.Montaje	10,00 €
17.Precio unitario material	**208,52 €**

Tabla 6-4 Precio unitario material

6.6 PRESUPUESTO FINAL

	Importe
1.Precio unitario material	208,52 €
2.Analizador Lógico	15,28 €
3.Osciloscopio	78,12 €
4.Licencias de Software	21.630,00€
5.Programación y diseño	22.900,00 €
Total	**44.831,92 €**
Beneficio Industrial (10%)	4.483,20€
Total sin IVA	**49.315,11€**
IVA (21%)	10.356,17 €
Total con IVA	**59.671,30 €**
Asciende el presupuesto final a la expresada cantidad de *CIQUENTA Y NIEVE MIL SEISCIENTOS SETENTA Y UN EUROS Y TREINTA CÉNTIMOS*	

Notas al presupuesto:

- Este presupuesto se ha elaborado para una unidad de prototipo de robot a dos ruedas y no contempla, por tanto, descuentos por volúmenes de compra.
- Los precios indicados corresponden a tarifas comerciales vigentes correspondientes a agosto de 2020. En el supuesto de que transcurrieran 12 meses o más en su desarrollo, estos precios deberán ser revisados.

Logroño a 5 de agosto de 2020

Firmado:

ROOSBEL VINICIO GUAYA VEGA, TITULADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA